Amrita Vishwa Vidyapeetham
Amrita School of Computing, Coimbatore
B.Tech Evaluation 1– -July 2025
Third Semester
Computer Science & Engineering
23CSE203 Data Structures & Algorithms

Duration: 30 minutes+ 4 days Date: 23-July-2025 Maximum: 2x10=20 Marks

Course Outcomes (COs):

CO	Course Outcomes
CO01	Understand the concept and functionalities of Data Structures and be able to implement them efficiently
CO02	Identify and apply appropriate data structures and their libraries to solve problems and improve their efficiency
CO03	Analyze the complexity of data structures and associated algorithms
CO04	Analyze the impact of various implementation and design choices on the data structure performance

Instructions:

1) READ ALL THE INSTRUCTIONS CAREFULLY!
2) Kindly read all the questions carefully & submit the answer files properly
3) The assessment has 2 parts, a written assessment for 30 minutes (which is submitted to the faculty) and s report submission to be done before 28-July-2025, 08:45AM .
4) LATE SUBMISSIONS WILL NOT BE ACCEPTED FOR ANY REASON!
5) Written Assessment: [CO2, BTL2, DL2]
    a) Write your COMPLETE Roll number & date in the TOP RIGHT corner of the paper (Roll No format. CB.SC.U4CSE24XXX, where XXX is the last 3 digits unique to the student)
    b) Students are not allowed to use any additional resources for this part
    c) For each of the questions given, the student will write the following on one face/side of a single sheet of A4 paper. No additional sheets are to be used.
        i) Explain your logic towards the design of your solution.
        ii) Mention the most appropriate data structure that can be used to represent this scenario as part of your explanation.
        iii) Mention the asymptotic time complexity of your solution with a ONE – TWO LINE justification.

6) Report [CO4, BTL4, DL2]:
    a) Deadline for report submission: 28-July-08:45AM
    b) A clear but not lengthy report comprising the following

        i) Initial solution pseudocode (the written part as a picture)
        ii) AI Usage Declaration: Mention one of the following that best describes your use of any AI agents/tools.
            (1) Did not use AI at any point of the assessment
            (2) Used AI as a search engine for general brainstorming
            (3) Used AI to check if my solution is correct
            (4) Used AI to generate solution which I validated
            (5) AI developed everything and I trust the solution`
        iii) What is the time complexity of the solution
        iv) Justification of time complexity.
        v) Final Solution pseudocode.
        vi) How did it evolve from initial to final solution. Describe your usage of AI/Web/other tools that empower you to reach the solution.
        vii) Appropriate (at least 3) test cases for your proposed solution to showcase the complexity of implemented data structure and associated algorithms
            (1) Resubmitting the sample test case will result in negative score.
            (2) Submitting minor modifications of the same sample test case will also result in negative scores.
        viii) Final code
            (1) Code should be in C++, JAVA or Python
            (2) Should accept inputs and display outputs as shared in driver code.
            (3) Must have a preamble as code comments that describe the working and logic of the code
            (4) Class/Method/Variable names should be meaningful and should not be random characters.
    c) The report should be submitted only as a PDF.
    d) Upload the report on Turnitin. The link will be shared later.
7) Implementation [CO1, BTL3, DL1]:
    a) Submit your final code (same as submitted in report) on github – link will be shared soon.
    b) Your implementation should be optimal in terms of both time & space usage
    c) Note that a driver code is available (in JAVA & Python)
8) **FAILURE TO FOLLOW THESE INSTRUCTIONS WILL RESULT IN LOSS OF MARKS.**
9) **Submissions showing Turnitin plagiarism levels up to 20% will be considered for the assessment; any values over 20 will result in the submission being ignored and not be awarded a score greater than zero.**

Question 1:

SCENARIO:

It's exam week, and you're spiraling. Your study plan is falling apart. To make things worse, your inner anime protagonist has awakened — and it refuses to let you study unless the conditions are "worthy of a true training arc."

So, what do you do?

You are forced to create an “Exam Survival App” — a dynamic task manager to balance your anime marathons and study sessions. This app can help you to juggle between:

- Watching episodes of your favorite anime,
- Eating ramen at oddly specific hours,
- Crying over syllabus PPTs, and
- Actually writing exams

Since your schedule keeps changing — like when your roommate suggests watching “Attack on Titan” finale at 3am — you need a system that supports:

a) Adding tasks dynamically (anime or study),
b) Removing tasks you rage-quit,
c) Viewing your current "training arc" in order or in reverse, depending on mood.

Your application should support the following operations:

Sno.	**Operation**	**Meaning**
1	A TaskID TaskName Type	Add a new task (Type = Anime or Study)
2	R TaskID	Remove a task by its ID
3	P	Print the current task arc (in order)
4	REV	Print the arc in reverse (just for emotional effect)

INPUT FORMAT:

- The first line contains a single integer n — the number of operations to be performed.
- The next n lines each contain one operation in one of the following formats:
    - *A TaskID TaskName TaskType*
        - Add a task with a unique ID, a task name (no spaces), and a type (Anime or Study).
        - Note that ID, Name and Type are space separated
    - R TaskID
        - A space separated text to remove the task with the given TaskID.
    - P
        - Print the current task list in order of addition
    - REV
        - Print the current task list in reverse order.

OUTPUT FORMAT:

For each operation, print the appropriate result:

- Add Task - Task <TaskID> (<TaskName> - <TaskType>) added.
- Remove Task
    - If task found - Task <TaskID> removed.
    - If not found - Task <TaskID> not found.
- Print Task
    - While there are tasks - <TaskID> <TaskName> <TaskType> ... (one line per task)
    - If no tasks print - "Task list is empty"
- Print in Reverse:
    - Same as print, but in reverse

SAMPLE INPUT:

```
7
A T001 WatchNaruto Anime
A T002 StudyDS Study
A T003 WatchJJK Anime
P
R T002
A T004 CryOverSyllabus Study
REV
```

<u>SAMPLE OUTPUT:</u>

```
Task T001 (WatchNaruto - Anime) added.
Task T002 (StudyDS - Study) added.
Task T003 (WatchJJK - Anime) added.
T001 WatchNaruto Anime
T002 StudyDS Study
T003 WatchJJK Anime
Task T002 removed.
Task T004 (CryOverSyllabus - Study) added.
T004 CryOverSyllabus Study
T003 WatchJJK Anime
T001 WatchNaruto Anime
```

Question 2:

SCENARIO:

Your roommate has started trading cryptocurrency and is begging you to build a simple crypto wallet — "It's just a glorified task tracker with a cool name," they insist. To stop their constant nagging, you agreed to build a custom blockchain system as a personal log of your wins, fails, and confusing moments while learning Web3 and smart contracts.

But before you can dive into coin trading, you need a working blockchain core — your very own "Blockchain". From YouTube tutorials you have learned that a blockchain is a distributed, append-only ledger made up of blocks, where each block stores a record (like a transaction or log), an identifier, and a reference to the previous block. This chain structure ensures 'immutability and transparency', and is fundamental to technologies like Bitcoin, Ethereum, and NFTs. Although real blockchains are cryptographically secured and decentralized, before diving into wallets and tokens, you need to build the foundation: a working blockchain-like ledger that can store logs, be manipulated dynamically, and support basic queries and views. No mining. No cryptography. Just data structures.

Your Task: Implement a Simplified Blockchain Log System. A blockchain is a secure, append-only list of records called blocks, each containing data, an identifier, and a classification. Real blockchains are cryptographically linked and decentralized. In this simulation, however, you're building a simplified version using data structures that you have studied - capturing the idea of ordered, verifiable logs.
In your implementation, the *BlockID* is a Unique identifier (e.g., B101), *BlockData* a one-word string describing the data (e.g., HostelNFT), *BlockType* represents the status Either Success or Fail
Your roommate insists that your implementation must be perfect and has reached out to your data structures professor to ensure that you do the job! Your faculty has defined operations for this simplified custom blockchain and insists that your solution must support the following operations.

SNO	Operation	Description
1	A BlockID BlockData BlockType	Add a new block (BlockType = Success or Fail)
2	D BlockID	Remove a block with the given ID
3	P	Print the blockchain in order of addition
4	R	Print the blockchain in reverse
5	S BlockType	Sort the blockchain with all blocks of the given type (Success or Fail) appearing first
6	F BlockID	Find and print the block with the given ID

To make matters worse, and has reached out to your data structures professor to ensure that you do the task properly

INPUT FORMAT:

- The first line contains an integer n — the number of operations.
- The next n lines each contain one operation, formatted as one of
  - A BlockID BlockData BlockType
  - D BlockID
  - P
  - R
  - S BlockType
  - F BlockID

OUTPUT FORMAT:

For each operation print the corresponding output as shown in the table

SNO	Operation	Description
1	A BlockID BlockData BlockType	Block <BlockID> (<BlockData> - <BlockType>) added.
2	D BlockID	If found: Block <BlockID> removed. If not: Block <BlockID> not found.
3	P	One line per block: <BlockID> <BlockData <BlockType> If empty: Blockchain is empty
4	R	Same as P but data and ID will be in reverse
5	S BlockType	Blockchain sorted with <BlockType> blocks first.
6	F BlockID	If found: <BlockID> <BlockData> <BlockType> If not: Block <BlockID> not found.

<u>SAMPLE INPUT:</u>

```
10
A B101 SubmittedAssignment Success
A B102 ForgotRoomKey Fail
A B103 DidNotStudyForTest Fail
F B102
S Success
P
R
D B104
D B103
P
```

<u>SAMPLE OUTPUT</u>

```
Block B101 (SubmittedAssignment - Success) added.
Block B102 (ForgotRoomKey - Fail) added.
Block B103 (DidNotStudyForTest - Fail) added.
B102 ForgotRoomKey Fail
Blockchain sorted with Success blocks first.
B101 SubmittedAssignment Success
B102 ForgotRoomKey Fail
B103 DidNotStudyForTest Fail
B103 DidNotStudyForTest Fail
B102 ForgotRoomKey Fail
B101 SubmittedAssignment Success
Block B104 not found.
Block B103 removed.
B101 SubmittedAssignment Success
B102 ForgotRoomKey Fail
```

Evaluation Rubrics for each question:

1. Initial written submission (3 marks) [CO2, BTL2, DL2]:
    a. Logic/Explanation – 2 marks
    b. Time complexity with justification – 1 mark
2. Implementation (1 mark) [CO1, BTL3, DL1]:
    a. Code successfully passes instructor test case – 1 mark
    b. Otherwise – 0 marks
3. Report (6 Marks) [CO4, BTL4, DL2]:
    a. Detailed and original description of the solution evolution – 2 marks
    b. Test case Generation – 1
    c. Time complexity analysis – 2 marks
    d. AI Usage Declaration – 1 mark