Oral Exam Prep Activity: Pair and Prepare

Oral Exam Prep Activity: Student Handout

Overview

Today, you will work with a partner to practice for your oral exam. You will explain concepts, design solutions, and answer quick questions. Use the rubrics provided to give each other helpful feedback.

Activity 1: Concept Talk (15 minutes)

- Each student selects **one topic** below.
- Partner listens carefully and fills out the rubric.
- After both students present, **debrief** by giving each other feedback.

Sample Topics:

1. Explain the advantages of separating interface and implementation when designing an ADT.
2. Explain how the doubling strategy in dynamic resizing works in array-based data structures in terms of amortized constant performance.
3. If your Array2D allowed external code to modify its internal Array or Array's NumPy array directly, what problems might arise? How does abstraction help avoid these issues?
4. Explain how using an array of buckets combined with linked-list chaining helps a HashMap handle collisions.
5. Compare and contrast stacks and queues in terms of their access patterns, typical use cases, and underlying data structure choices. Why would you choose one over the other in a given problem?

Activity 2: Design Talk (15 minutes)

- Work together to design a solution for one topic below.
- After designing, ask each other a **follow-up question** to deepen understanding.

Sample Topics:

1. Propose an alternate design for Array2D where rows can have different lengths, but the structure still uses our Array as the underlying data structure.
2. How would you implement the `__getitem__` and `__setitem__` methods for our LinkedList class to support indexing?
3. How would we modify our CircularQueue design to support Deque operations?
4. How would you implement the **sorted** method for a HashMap that returns an Array of key/value pairs sorted by key?

Activity 3: Rapid Fire Reflect and Swap (15 minutes)

- Decide who goes first.
- Take turns answering each question below, alternating back and forth.
- Record your feedback in the rubric.
- After all questions are answered, **debrief** and discuss your strengths and areas to practice.

Sample Rapid Fire Questions:

1. When would a fixed-size array be preferable over a dynamic array?
2. What is the difference between a shallow copy and a deep copy?
3. How does Python's list differ from a true array?
4. What is the time complexity of accessing an element in a linked list?
5. Why is modular arithmetic useful in implementing circular structures?
6. Explain the difference between a stack and a queue in terms of their access patterns.
7. How could you use Array2D to support a game like Pac-Man that requires wrapping around the edges?
8. What would be a disadvantage of using an array-based Deque?
9. What is the purpose of the `__repr__` method in a class?
10. What happens if you forget to update both `next` and `prev` pointers in a doubly linked list?

Rubrics

Concept Talk Rubric

Presenter Name	Topic	Clear Explanation (1-5)	Relevant Example Given (1-5)	Confident Speaking (1-5)	Listener Notes

Rubric Descriptions:

- **Clear Explanation:** Did the explanation stay focused and understandable?
- **Relevant Example Given:** Was a helpful and relevant example used?
- **Confident Speaking:** Did the presenter sound confident and well-prepared?

Design Talk Checklist

Evaluate your design based on the following criteria as guidelines:

1. Clearly rephrase or clarify the design problem in your own words before starting.
2. State current assumptions based on working knowledge of the current data structure design.
3. Clearly state the required operations and how they will be handled in the new design.
4. Discuss how the modification improves the current design.
5. Discuss assumptions made in the new design.
6. Discuss edge cases to consider.
7. Discuss potential trade-offs and limitations of the new design.
8. Discuss time/space complexity awareness of the design.

Rapid Fire Reflect and Swap Rubric

Question #	Partner Answering	Clear and Accurate (1-5)	Needed Clarification (Y/N)	Listener Notes

Rubric Descriptions: - **Clear and Accurate:** Was the answer correct and easy to understand? - **Needed Clarification:** Did the listener need to ask for clarification or did the speaker answer it completely?

Sample Answers for Oral Exam Prep Activities

Activity 1: Concept Talk Sample Answers

1. **Advantages of separating interface and implementation:** It hides implementation details, allowing flexibility to change internals without affecting user code. Makes code modular, easier to maintain.
2. **Doubling strategy and amortized performance:** Although resizing an array is expensive, it happens rarely. Over many insertions, the average cost per insertion remains O(1).
3. **Problems with exposing internal arrays in Array2D:** External code could corrupt the grid, cause invalid accesses, or break invariants. Abstraction protects structure integrity.
4. **Array of buckets with chaining in HashMap:** Handles collisions by allowing multiple elements to exist in the same bucket, linked together. Maintains average constant lookup even with collisions.
5. **Stack vs Queue comparison:** Stack is LIFO (Last In First Out), Queue is FIFO (First In First Out). Stack is used in backtracking, Queue in scheduling or breadth-first search.

Activity 2: Design Talk Sample Answers (Verbose with Code)

1. Alternate Array2D with Variable Row Lengths

Design Idea: Use an Array of Arrays, where each row is a separate dynamic Array. Handle row-based indexing carefully.

```
class Array2D:
    def __init__(self, rows):
        self._rows = Array(rows)
        for i in range(rows):
            self._rows[i] = Array()  # Empty row initially
```

---

2. Implement `__getitem__` and `__setitem__` for LinkedList

Design Idea: Traverse node-by-node counting indices until reaching the target.

```
class LinkedList:
    def __getitem__(self, index):
        current = self._head
        for _ in range(index):
            if current is None:
                raise IndexError("Index out of range")
            current = current.next
        return current.data

    def __setitem__(self, index, value):
        current = self._head
        for _ in range(index):
            if current is None:
                raise IndexError("Index out of range")
            current = current.next
        current.data = value
```

---

3. Modify CircularQueue to Support Deque Operations

Design Idea: Allow insertion and deletion at both head and tail; use modular arithmetic to wrap indices.

```
class CircularDeque:
    def add_front(self, item):
        self._head = (self._head - 1) % len(self._data)
        self._data[self._head] = item

    def add_rear(self, item):
        self._data[self._tail] = item
        self._tail = (self._tail + 1) % len(self._data)

    def remove_front(self):
        item = self._data[self._head]
        self._head = (self._head + 1) % len(self._data)
        return item
```

```
    def remove_rear(self):
        self._tail = (self._tail - 1) % len(self._data)
        item = self._data[self._tail]
        return item
```

---

4. Implement `__sorted__` Method for HashMap

Design Idea: Collect all key-value pairs into an array and sort by key.

```
class HashMap:
    def __sorted__(self):
        pairs = []
        for bucket in self._buckets:
            if bucket:
                for (key, value) in bucket:
                    pairs.append((key, value))
        return sorted(pairs, key=lambda pair: pair[0])
```

Activity 3: Rapid Fire Sample Answers

1. **Fixed-size array preferable:** When size is known ahead of time; better memory predictability.
2. **Shallow vs deep copy:** Shallow copy copies references; deep copy copies full objects recursively.
3. **Python list vs true array:** Python lists are dynamic, heterogeneous; true arrays are fixed-size, homogeneous.
4. **Linked list element access:** O(n) time since it requires traversal.
5. **Modular arithmetic in circular structures:** Allows wrap-around behavior when indexing.
6. **Stack vs Queue access patterns:** Stack = LIFO, Queue = FIFO.
7. **Array2D for Pac-Man:** Use modular indexing to wrap around edges.
8. **Disadvantage of array-based deque:** Inefficient front insertions and removals unless using circular buffer.
9. **Purpose of `__repr__`:** Provides a readable string representation of an object for debugging.

10. **Forgetting to update `next` and `prev`:** Causes broken links in the list, leading to traversal errors.