JOHN ELMER BLANQUISCO BOBIER JR & VINCENT MACAUYAM

Scenario: Galactic Cargo Management

The program places you in a futuristic setting where you manage cargo shipments across the galaxy.

Story

In the far future, humans have built colonies on planets across the galaxy. These colonies depend on shipments of important supplies like food, medicine, and tools to survive and grow. You are the cargo manager on the spaceship *SS Nexus*, one of the fastest and most advanced freighters in the galaxy.

Your job is to:

1. **Insert Shipments:** Add new shipments into the priority system without disrupting the order.
2. **Dispatch Shipments:** Remove and deliver the most urgent shipments efficiently.
3. **Maintain Order:** Use the Heapify process to keep the system working correctly.
4. **Monitor Changes:** Watch how your actions impact the priority structure and make adjustments as needed.

The backstory immerses you as a "cargo manager" on a spaceship, adding a narrative flair. Print statements like *“Inserted shipment with priority X into the heap”* create engaging feedback.

Game Logic

The program uses a Max-Heap to prioritize shipments based on their urgency.

- **Heap Operations:** Demonstrates insertion, removal (dispatch), and heapify operations in a Max-Heap.
- **Heapify Functions:** Maintains heap properties using `heapifyUp` and `heapifyDown`.
- **Practical Use Case:** Illustrates managing priorities in real-world scenarios using heaps.
- **Visual Representation:** Displays the heap as a vector using the `displayHeap` function to show its current state.

Task Operations

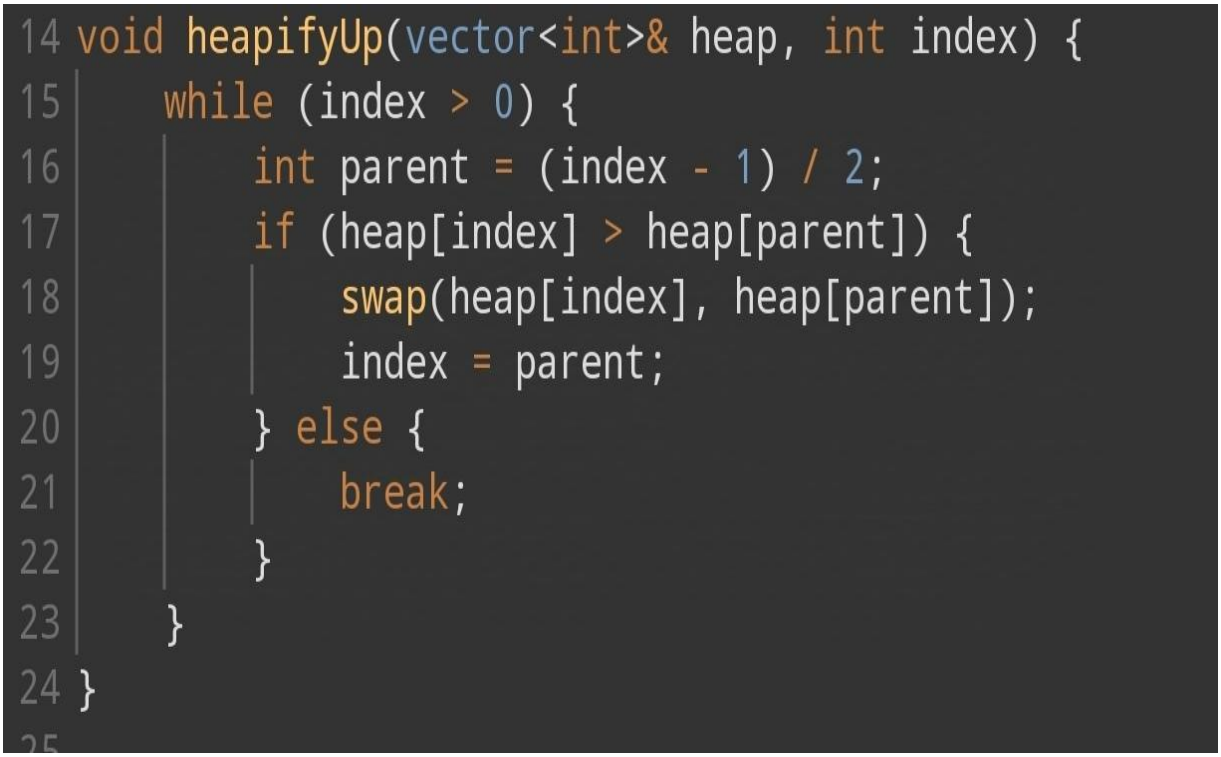
The program provides three primary tasks:

1. **Insert Shipments:** Add a shipment with a given priority into the Max-Heap without breaking the heap structure.
2. **Dispatch Shipments:** Remove the highest-priority shipment (root) from the heap while maintaining the Max-Heap property.
3. **Monitor Heap State:** View the current state of the heap after each operation to visualize the changes.

Program Code Explanation

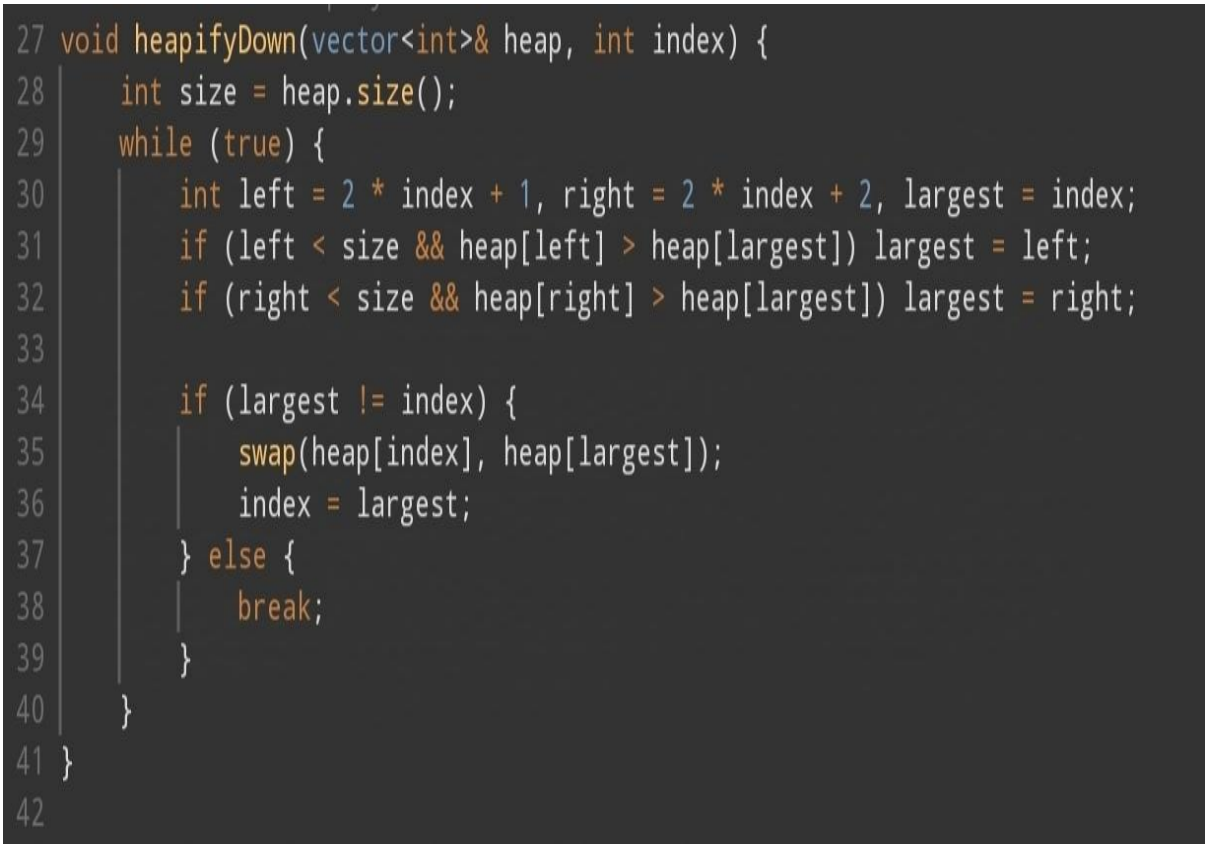
Heap Functions:

1. **`heapifyUp`:** Ensures the heap property when an element is added (bubbles up the new element).



```
14 void heapifyUp(vector<int>& heap, int index) {
15     while (index > 0) {
16         int parent = (index - 1) / 2;
17         if (heap[index] > heap[parent]) {
18             swap(heap[index], heap[parent]);
19             index = parent;
20         } else {
21             break;
22         }
23     }
24 }
```

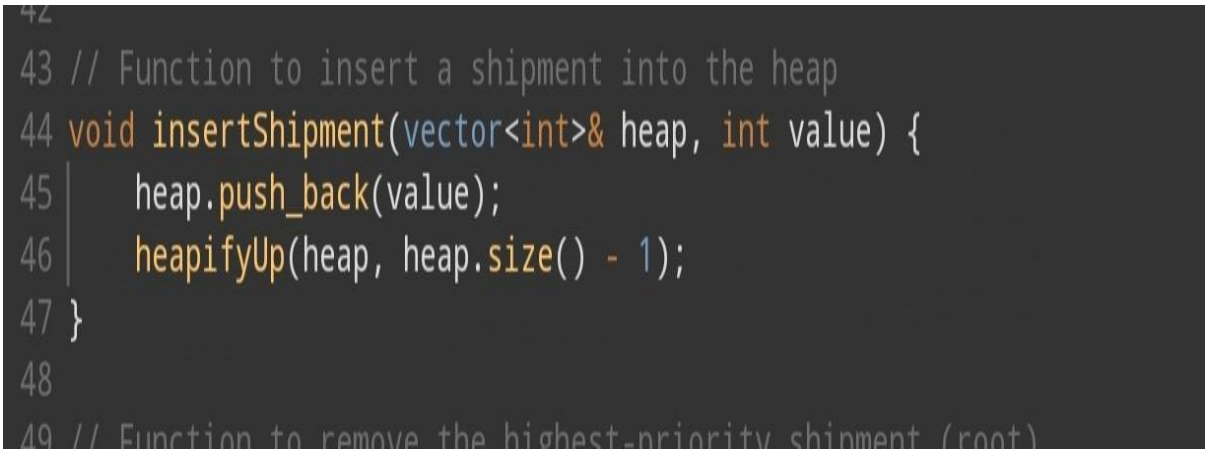
2. **`heapifyDown`:** Restores the heap property after the root is removed (pushes down the root to its correct position).



```
27 void heapifyDown(vector<int>& heap, int index) {
28     int size = heap.size();
29     while (true) {
30         int left = 2 * index + 1, right = 2 * index + 2, largest = index;
31         if (left < size && heap[left] > heap[largest]) largest = left;
32         if (right < size && heap[right] > heap[largest]) largest = right;
33 
34         if (largest != index) {
35             swap(heap[index], heap[largest]);
36             index = largest;
37         } else {
38             break;
39         }
40     }
41 }
42
```

3.**. Insert a Shipment**

This function adds a new shipment to the heap and restores the heap property using `heapifyUp`.



```
43 // Function to insert a shipment into the heap
44 void insertShipment(vector<int>& heap, int value) {
45     heap.push_back(value);
46     heapifyUp(heap, heap.size() - 1);
47 }
48 
49 // Function to remove the highest-priority shipment (root)
```

4.**Remove the Highest-Priority Shipment**

This function removes the root (highest priority) shipment and restores the heap property using `heapifyDown`.

```
50 int removeShipment(vector<int>& heap) {
51     if (heap.empty()) return -1;
52
53     int root = heap[0];
54     heap[0] = heap.back();
55     heap.pop_back();
56     if (!heap.empty()) heapifyDown(heap, 0);
57
58     return root;
59 }
```

5.**Display the Heap**

This function simply prints the heap in its current state

```
5
6 // Function to display the heap as a vector
7 void displayHeap(const vector<int>& heap) {
8     cout << "[ ";
9     for (int val : heap) cout << val << " ";
10    cout << "]\n";
11 }
12
```

Program Logic:

- The program uses a loop where users can perform actions like inserting, dispatching, or viewing the heap.
- A `vector` represents the heap, while predefined `shipments` provide initial values for demonstration.

Interactive Features:

- Users select from available shipments, ensuring that only valid priorities are added.
- The program ends gracefully when all shipments are managed, giving closure to the scenario.

Instructions for the User

The program includes user-friendly commands to ensure an interactive and enjoyable experience:

Commands:

1. **Insert Shipments:** Select a shipment by its priority value to add it to the heap.
2. **Dispatch Shipments:** Remove and deliver the highest-priority shipment.
3. **View the Heap:** Display the current state of the heap to observe changes.
4. **Exit:** Close the program when all shipments have been managed.

Guided Inputs:

- The program prompts you to select a shipment's priority value from the available list.
- Validates input to ensure only appropriate values are used.

Examples:

1. **Insertion:** Adds a shipment to the heap while showing the updated heap.
2. **Dispatch:** Removes the highest-priority shipment and displays the remaining heap.

<u>This is the Steps / Instructions we follow in Creating our Game:</u>

Step 1: Form a Partnership

Work with a partner to complete this activity.

Discuss and agree on a unique theme or scenario for your challenge.

Step 2: Set the Scene

Imagine a creative scenario or theme for your assessment.

"Galactic Cargo Management"

- In the far future, humans have built colonies on planets across the galaxy. These colonies depend on shipments of important supplies like food, medicine, and tools to survive and grow. To keep everything running smoothly, a system is needed to organize and prioritize deliveries based on how urgent they are.
- participants are on a spaceship named SS Nexus, one of the fastest and most advanced freighters in the galaxy. participants will take on the role of a "Galactic Resource Manager" for a futuristic space colony. The colony's survival depends on efficiently managing shipments of rare resources from different planets. Each resource has a priority based on its value. The challenge involves using Heap Data Structures to prioritize, manage, and allocate resources dynamically as new shipments arrive or existing resources are consumed.

Step 3: Define Your Learning Goals

Together, decide what the participants should learn or demonstrate by completing your tasks.

- Mastering insertion and deletion in Max-Heaps or Min-Heaps:
- Adding shipments into the priority system (Max-Heap or Min-Heap) corresponds to the insertion process.
- Dispatching the most urgent shipment aligns with deleting the root (highest-priority element) in a heap.

Implementing the Heapify operation on an array:

- The Heapify process is used to fix the heap whenever shipments are added, removed, or the order is disrupted.
- This ensures the priority system always follows the heap property.
- Converting between heap types or visualizing heaps:
- Visualizing the heap structure through the display helps participants understand the hierarchy and how operations affect it.
- You could optionally add a feature to convert between Max-Heap and Min-Heap (e.g., changing how priorities are interpreted).

Step 4: Create the Tasks

Include at least three tasks related to heaps.

Insert elements into a heap: Users add shipments (with priorities) into the heap, and the system displays the updated heap after each insertion.

Implement a function to delete the root and restore the heap property: The function removes the root (highest-priority shipment) and restores the heap property by rebuilding the heap.

Write a program to Heapify a random array into a Max-Heap: This is covered by the buildHeap function in your code, which uses heapify to transform an unsorted array into a valid Max-Heap.