Data Structures and Algorithms
SE-F22
LAB-12

Issue Date: May 24, 2024

Start Time: 09:45 AM
Completion Time: 12:45 PM

Total Marks: 50

Objective:

In this lab, students will learn how to implement a priority queue using a max-heap structure. They'll explore the significance of prioritizing tasks, such as in operating system scheduling, and gain practical experience by coding enqueue, dequeue, and peek operations.

Instructions:

1) Follow the question instructions very carefully, no changes in function prototypes are allowed.
2) You could solve the following growth functions on paper.
3) Anyone caught in an act of plagiarism would be awarded an “F” grade in this Lab.

Task Scheduling in an Operating System Lab

In an operating system, multiple tasks or processes need to be scheduled for execution. Each task is assigned a priority level, which determines the order in which the tasks should be executed. Higher priority tasks should be executed before lower priority ones. If two tasks have the same priority, they can be handled in the order they arrive.

Task 01 (Scheduler) [25 Marks]

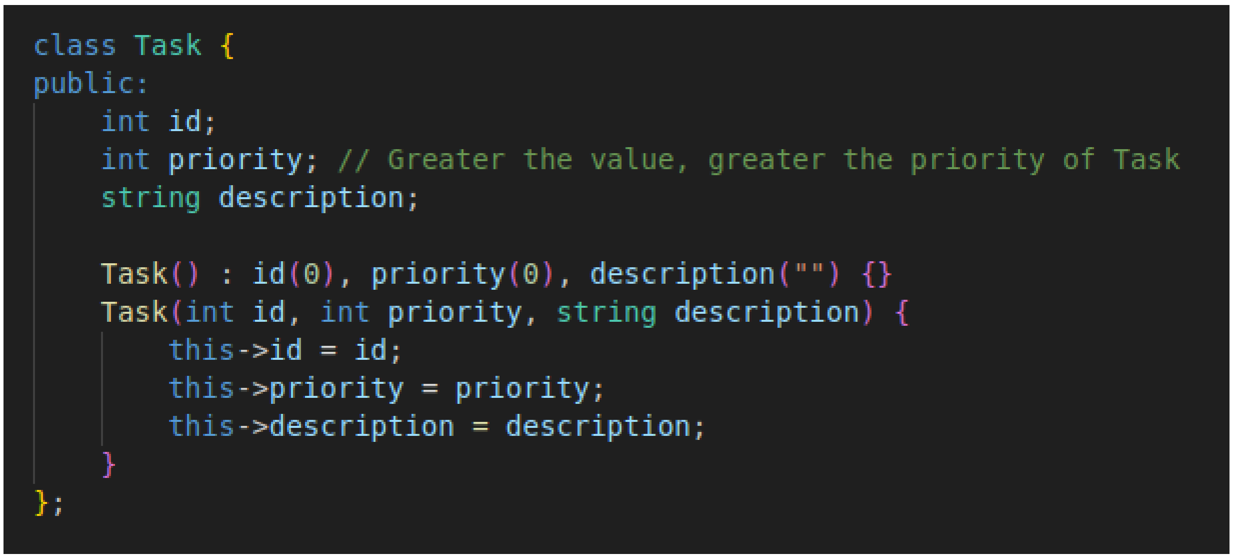
Implement a priority queue using a heap to manage the scheduling of tasks based on their priority levels.

Task Definition:

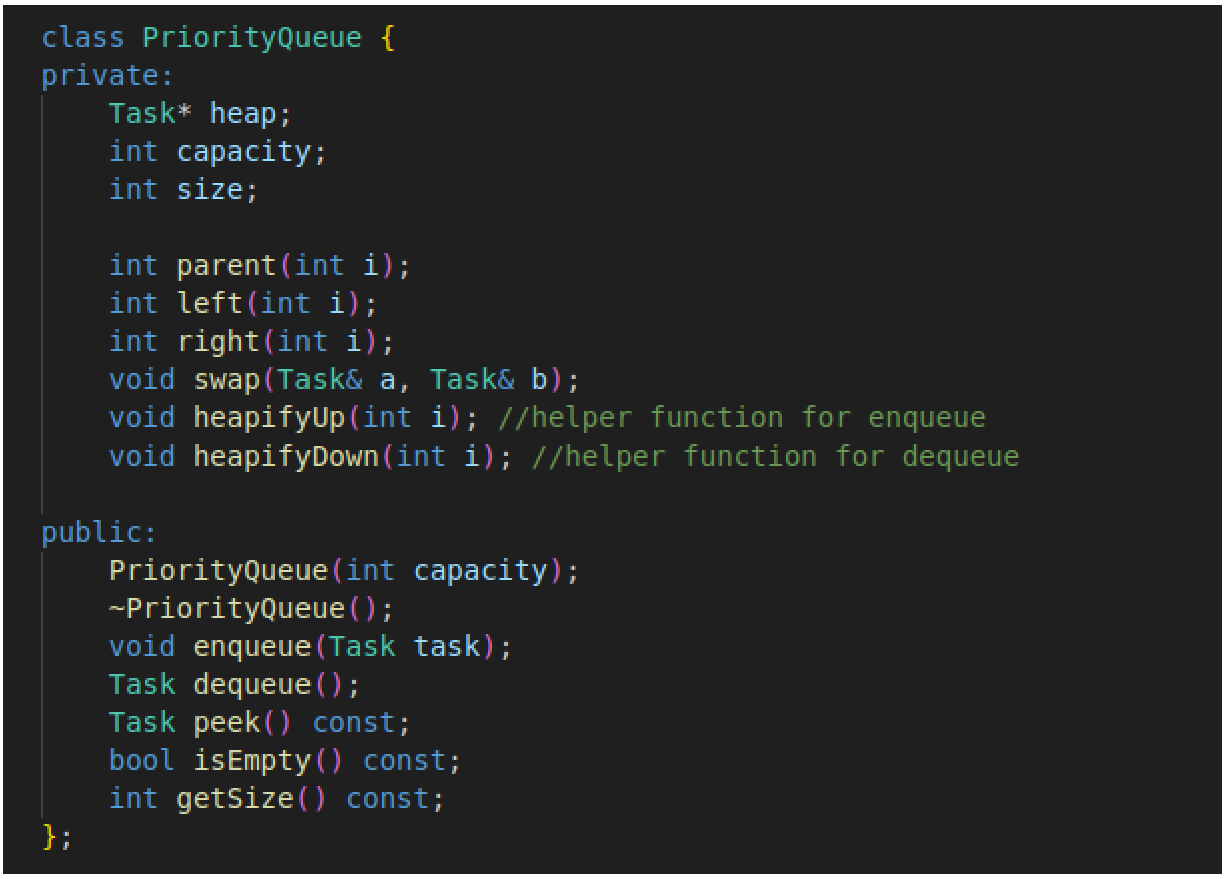
- Each task has a unique identifier (ID), a priority level, and a description.
- For simplicity, let's assume priorities are integers where a higher number indicates a higher priority.

Priority Queue Operations:

- **Enqueue (Insert Task)**: Add a new task to the queue with a given priority.
- **Dequeue (Remove Highest Priority Task)**: Remove and return the task with the highest priority.
- **Peek (View Highest Priority Task)**: View the task with the highest priority without removing it from the queue.



```cpp
class Task {
public:
    int id;
    int priority; // Greater the value, greater the priority of Task
    string description;

    Task() : id(0), priority(0), description("") {}
    Task(int id, int priority, string description) {
        this->id = id;
        this->priority = priority;
        this->description = description;
    }
};
```

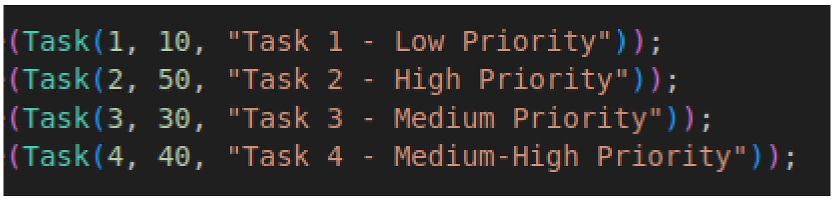


```cpp
class PriorityQueue {
private:
    Task* heap;
    int capacity;
    int size;

    int parent(int i);
    int left(int i);
    int right(int i);
    void swap(Task& a, Task& b);
    void heapifyUp(int i); //helper function for enqueue
    void heapifyDown(int i); //helper function for dequeue

public:
    PriorityQueue(int capacity);
    ~PriorityQueue();
    void enqueue(Task task);
    Task dequeue();
    Task peek() const;
    bool isEmpty() const;
    int getSize() const;
};
```

Task 02 (Run Simulation) [5 Marks]

Create a priority queue, add some tasks to the queue and perform operations on the queue to verify the correctness of the implementation.

Sample Run:

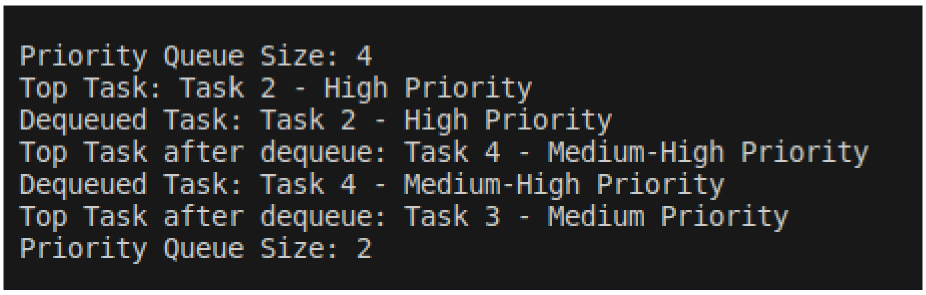


```
(Task(1, 10, "Task 1 - Low Priority"));
(Task(2, 50, "Task 2 - High Priority"));
(Task(3, 30, "Task 3 - Medium Priority"));
(Task(4, 40, "Task 4 - Medium-High Priority"));
```

Operations:

peek()
dequeue()
peek()
dequeue()
peek()

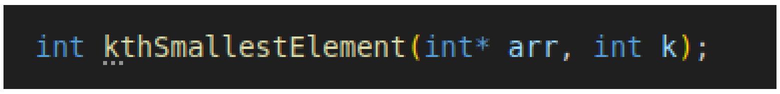
Output:



```
Priority Queue Size: 4
Top Task: Task 2 - High Priority
Dequeued Task: Task 2 - High Priority
Top Task after dequeue: Task 4 - Medium-High Priority
Dequeued Task: Task 4 - Medium-High Priority
Top Task after dequeue: Task 3 - Medium Priority
Priority Queue Size: 2
```

Task 03 (k-th element in array using min-heap) [20 Marks]

In this task you are required to implement a min-heap that would be used to find k-th smallest element from the input array.



```
int kthSmallestElement(int* arr, int k);
```

Sample Run:

Input Array: [2,3,9,4,0,8]
k: 4
Output: 4