PSUDOCODE FOR QUEUE

```
class CustomQueue<T>:
    private Object[] queue
    private int front
    private int rear
    private int size
    private static final int DEFAULT_CAPACITY = 10

    function CustomQueue():
        queue = new Object[DEFAULT_CAPACITY]
        front = 0
        rear = -1
        size = 0

    function ensureCapacity():
        if size == queue.length:
            newQueue = new Object[size * 2]
            copyElements(queue, newQueue, front, size - front)
            copyElements(queue, newQueue, 0, rear + 1)
            queue = newQueue
            front = 0
            rear = size - 1

    function enqueue(element: T):
        ensureCapacity()
        rear = (rear + 1) % queue.length
        queue[rear] = element
        size++

    function dequeue(): T:
        if not isEmpty():
            element = queue[front]
            front = (front + 1) % queue.length
            size--
            return element
        return null

    function isEmpty(): boolean:
        return size == 0

    function size(): int:
        return size

    function copy(): CustomQueue<T>:
        copyQueue = new CustomQueue<T>()
        for i = 0 to size - 1:
            copyQueue.enqueue(queue[(front + i) % queue.length])
        return copyQueue

    function countOccurrences(element: T): int:
        count = 0
        for i = 0 to size - 1:
            if element.equals(queue[(front + i) % queue.length]):
```

```
            count++
        return count

    function findMostFrequent(): T:
        if isEmpty():
            return null

        mostFrequent = null
        maxCount = 0

        for i = 0 to size - 1:
            item = queue[(front + i) % queue.length]
            count = countOccurrences(item)
            if count > maxCount:
                maxCount = count
                mostFrequent = item

        return mostFrequent

    function findLongest(): T:
        if isEmpty():
            return null

        longest = queue[front]

        for i = 0 to size - 1:
            item = queue[(front + i) % queue.length]
            if length(item.toString()) > length(longest.toString()):
                longest = item

        return longest

class QueueTextParser:
    queue = new CustomQueue<String>()

    function readTextFile(filePath: String): String:
        fileContent = ""
        try:
            open and read file at filePath
            while line is not null:
                append line to fileContent
        catch IOException:
            print error details
        return fileContent

    function readAndAddToQueue():
        filePath = "/Users/egj/Desktop/Queue/HouseofUsher.txt"
        text = readTextFile(filePath)
        words = split text into words using non-alphanumeric characters
        for word in words:
            if word is not empty:
                queue.enqueue(toLowerCase(word))

    function main():
```

```
	textParser = new QueueTextParser()
	textParser.readAndAddToQueue()

	superhumanCount = textParser.getWordCount("superhuman")
	chiromancyCount = textParser.getWordCount("chiromancy")
	discernibleCount = textParser.getWordCount("discernible")
	percutaneousCount = textParser.getWordCount("percutaneous")
	unsatisfactoryCount = textParser.getWordCount("unsatisfactory")

	totalEntries = textParser.getSize()
	wordsOver20 = textParser.getWordsMoreThan20()
	mostFrequentWord = textParser.getMostFrequentWord()
	longestWord = textParser.getLongestWord()

	print "superhuman count:", superhumanCount
	print "chiromancy count:", chiromancyCount
	print "discernible count:", discernibleCount
	print "percutaneous count:", percutaneousCount
	print "unsatisfactory count:", unsatisfactoryCount

	print "Total entries:", totalEntries
	print "Words over 20 in the first 1000 entries:", wordsOver20
	print "Most frequent word:", mostFrequentWord
	print "Longest word:", longestWord

	wordsToFind = ["superhuman", "chiromancy", "unsatisfactory", "percutaneous",
"discernible"]
	textParser.countDequeueOperationsForWords(wordsToFind)

	print "PROCESSES COMPLETED"
```