### 文件 LinkPriorityQueue.h 中的函数作用：

1. **LinkPriorityQueue()**: 构造函数，初始化优先级队列。
2. **LinkPriorityQueue(const LinkPriorityQueue& other)**: 拷贝构造函数，用于深拷贝另一个优先级队列。
3. **~LinkPriorityQueue()**: 析构函数，释放优先级队列的内存。
4. **Push(DataType x, int priority)**: 向优先级队列中插入带有优先级的数据元素。
5. **Pop()**: 弹出并返回优先级最高的元素。
6. **Get()**: 获取优先级最高的元素，但不弹出。
7. **Priority\_Get(int priority)**: 获取指定优先级的元素。
8. **ResetPriority(const DataType x, int priority)**: 重设某元素的优先级。
9. **Empty()**: 检查队列是否为空。

### 文件 LinkQueue.h 中的函数作用：

1. **LinkQueue()**: 构造函数，初始化链式队列。
2. **~LinkQueue()**: 析构函数，释放链式队列的内存。
3. **EnQueue(DataType x)**: 向队列末尾添加数据元素。
4. **DeQueue()**: 从队列头部删除并返回数据元素。
5. **GetQueue()**: 获取队列头部的元素但不删除。
6. **Empty()**: 检查队列是否为空。

### 文件 SeqStack.h 中的函数作用：

1. **SeqStack()**: 构造函数，初始化顺序栈。
2. **~SeqStack()**: 析构函数，销毁栈（这里未做具体实现）。
3. **Push(DataType x)**: 向栈中添加数据元素。
4. **Pop()**: 弹出并返回栈顶元素。
5. **GetTop()**: 获取栈顶元素但不弹出。
6. **Empty()**: 检查栈是否为空。

### 文件 LinkPriorityQueue.h 中的时间复杂度：

1. **LinkPriorityQueue()**: O(1)
2. **LinkPriorityQueue(const LinkPriorityQueue& other)**: O(n) （n为队列中元素个数）
3. **~LinkPriorityQueue()**: O(n) （销毁队列中的所有元素）
4. **Push(DataType x, int priority)**: O(n) （可能需要遍历链表插入元素）
5. **Pop()**: O(1) （直接弹出第一个元素）
6. **Get()**: O(1) （获取第一个元素）
7. **Priority\_Get(int priority)**: O(n) （需要遍历找到指定优先级的元素）
8. **ResetPriority(const DataType x, int priority)**: O(n) （需要遍历找到元素并更新其优先级）
9. **Empty()**: O(1)

### 文件 LinkQueue.h 中的时间复杂度：

1. **LinkQueue()**: O(1)
2. **~LinkQueue()**: O(n) （释放所有元素）
3. **EnQueue(DataType x)**: O(1) （添加元素到队列末尾）
4. **DeQueue()**: O(1) （删除队列头部元素）
5. **GetQueue()**: O(1) （获取头部元素）
6. **Empty()**: O(1)

### 文件 SeqStack.h 中的时间复杂度：

1. **SeqStack()**: O(1)
2. **~SeqStack()**: O(1)
3. **Push(DataType x)**: O(1) （在数组末尾添加元素）
4. **Pop()**: O(1) （弹出栈顶元素）
5. **GetTop()**: O(1) （获取栈顶元素）
6. **Empty()**: O(1)

这三个文件实现了优先级队列、链式队列和顺序栈的基本操作，每个函数的时间复杂度取决于其具体的数据结构和操作类型。