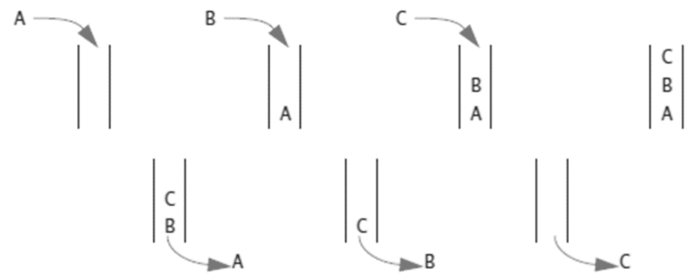
CSE2122 (Spring 2020)
In-class activity 1, 2/28/2020

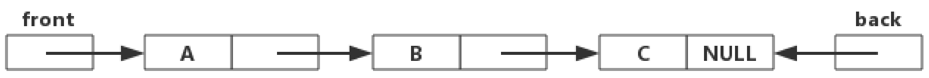
A queue is a data structure that handles data in a first-in/first-out (FIFO) fashion. A queue behaves exactly the same as a line of people waiting for a bank teller or other service. The people are served in the order they enter the line (the queue). The operation of a queue is diagrammed in the following figure.



We can define a `Queue` class as follows.

```
struct QueueNode{
    char data;
    QueueNode *link;
};

class Queue{
public:
    Queue();  // Initializes the object to an empty queue.
    Queue(const Queue& aQueue);
    ~Queue();

    //Postcondition: item has been added to the back of the queue.
    void add(char item);

    //Precondition: The queue is not empty.
    //Returns the item at the front of the queue and removes that
    //item from the queue.
    char remove();

    //Returns true if the queue is empty. Returns false otherwise.
    bool empty() const;

private:
    QueueNode *front; //Points to the head of a linked list.
    QueueNode *back;  //Points to the node at the other end of the queue
};
```

Unlike in the `Stack` class, here we use two pointers, `front` and `back`. If a `Queue` object represents an empty queue, both pointers will have the value `NULL`. If a `Queue` object represents a non-empty queue, the front pointer will store the memory address of the first node in the queue, and the back pointer will store the memory address of the last node in the queue. The following figure shows an example of a `Queue` object.



Besides the data members, several functions are defined for the class:

- Default constructor: initializes the queue to an empty queue.
- Copy constructor: initializes a new queue as a copy of an existing queue.
- Destructor: frees the memory that a `Queue` object may have acquired.
- `add`: adds an item to the back of the queue.
- `remove`: returns the item at the front of the queue and removes that item from the queue. If this queue is empty, this function will print a hint message.
- `empty`: returns true if the queue is empty, and returns false otherwise.

First, we will define the add function. Consider a Queue object `front -> A -> B <- back`. After `add('C')` is called to this object, it should be `front -> A -> B -> C <- back`. From this example, we can see that to complete this function, a new node needs to be created, and the `link` pointer of the old last node and the `back` pointer need to be changed. We can get the following segment of code as a part of the function body.

```
QueueNode *temp = new QueueNode;
temp->data = item;
temp->link = NULL;
back->link = temp;
back = temp;
```

Now think about this question: is this segment of code enough for all possible cases?

If we call this function to an empty queue named `empty`, the expected queue after the function call `empty.add('C')` is `front -> C <- back`. In the empty queue case, both `front` and `back` pointers need to be changed. We can get the following segment of code for empty queues:

```
front = new QueueNode;
front->data = item;
front->link = NULL;
back = front;
```

Combining these two parts, we can define the `add` function as:

```
void Queue::add(char item){
    if (empty()){  // for empty queues
        front = new QueueNode;
        front->data = item;
        front->link = NULL;
        back = front;
    }
    else{  // for non-empty queues
        QueueNode *temp;
        temp = new QueueNode;
        temp->data = item;
        temp->link = NULL;
        back->link = temp;
        back = temp;
    }
    return;
}
```

Make sure you understand how the `Queue` class is supposed to work before you continue.

You will be asked to complete two functions and answer one question in the following part.

1. Complete the copy constructor and the remove function. Other functions are provided.

```
Queue::Queue(){
    front = NULL;
    back = NULL;
}

Queue::~Queue(){
    while (!empty( ))
        remove( );
}

bool Queue::empty() const{
    return (back == NULL); //front == NULL would also work
}

char Queue::remove(){
    // your code here











}
```

```
}

Queue::Queue(const Queue& aQueue){
 // your code here





}
```

2. Consider the following definition. Does it work as the add function, why or why not?

```
void Queue::add(char item){
    QueueNode* temp;
    temp = new QueueNode;
    temp->data = item;
    temp->link = NULL;
    back->link = temp;
    back = temp;
    if (front==NULL)
        front = back;
    return;
}
```