Queues

Queues

- A stack is LIFO (Last-In First Out) structure.
- In contrast, a *queue* is a FIFO (First-In First-Out) structure.
- A queue is a linear structure for which items can be only inserted at one end and removed at another end.

Queues

- A real world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first.
- More real-world example can be seen as queues at ticket windows & bus-stops.

Terminologies

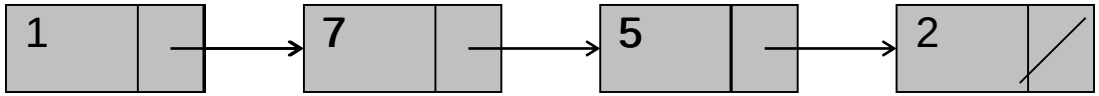
- Front - front/first element
- Rear - last element
- Enqueue - Insert an element at rear
- Dequeue - Delete an element from front

Queue Operations

Enqueue(X) –	place X at the *rear* of the queue.
Dequeue() --	remove the *front* element and return it.
Front() --	return front element without removing it.
IsEmpty() --	return TRUE if queue is empty, FALSE otherwise

Implementing Queue

Using linked List:

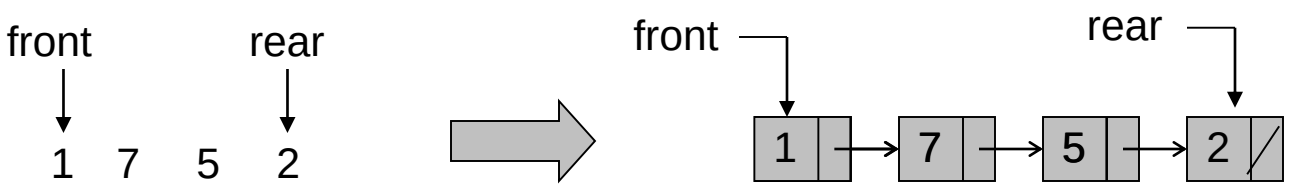


Implementing Queue

- Insert works in constant time for either end of a linked list.
- Remove works in constant time only.
- Seems best that head of the linked list be the front of the queue so that all removes will be from the front.
- Inserts will be at the end of the list.

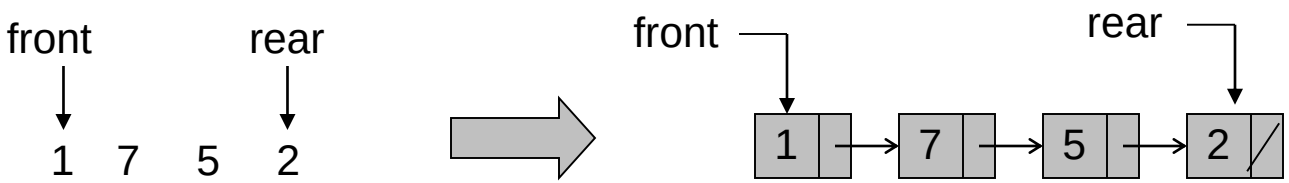
Implementing Queue

- Using linked List:

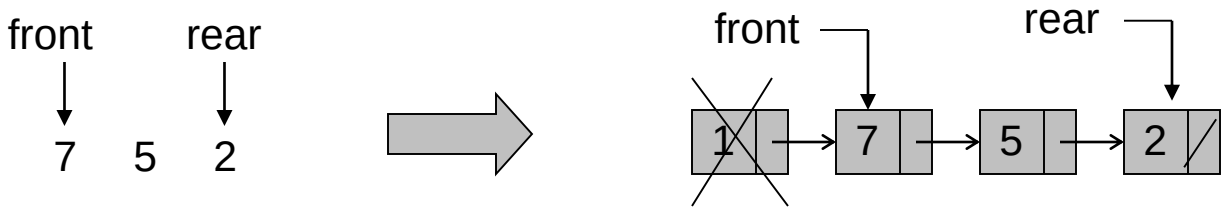


Implementing Queue

- Using linked List:

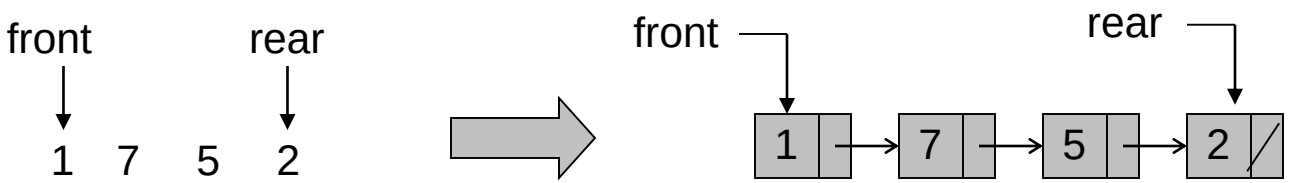


dequeue()

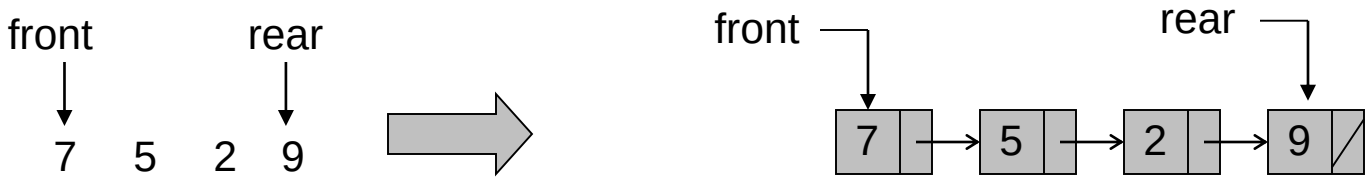


Implementing Queue

- Using linked List:



enqueue(9)



Implementing Queue (using linklist)

- You need at least four files to implement queue:
  1. Node.h (int object, Node* next)
  2. Queue.h
  3. Queue.cpp
  4. Main.cpp

Queue.h

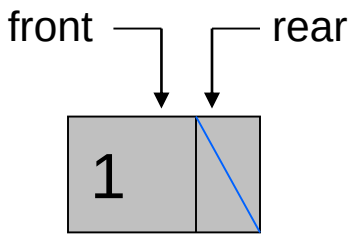
```
class Queue
{
        Node * front;   // headnode/firstnode
        Node * rear;    //lastnode
public:
        Queue();   //constructor
        void enqueuer(int x);
        int dequeue();
        int front();
        int isEmpty();
        ~Queue();
};
```

Queue.cpp (enqueue(1))

```
Void Queue::enqueue(int x){

    Node* newNode = new Node(); //line 1
    newNode->set(x);            //line 2
    newNode->setNext(nullptr);  //line 3
    if(front==nullptr){         //line 4

            front=rear=newnode;  //line 5
    }
    else{
            rear->setNext(newNode);  //line 6
            rear = newNode;          //line 7
      }
}
```

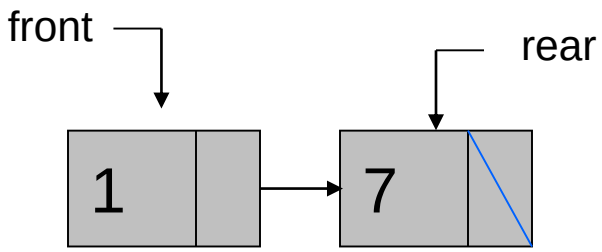


Queue.cpp (enqueue(7))

```
Void Queue::enqueue(int x){

    Node* newNode = new Node(); //line 1
    newNode->set(x);            //line 2
    newNode->setNext(nullptr);  //line 3
    if(front==nullptr){         //line 4

            front=rear=newnode;  //line 5
    }
    else{
            rear->setNext(newNode);  //line 6
            rear = newNode;          //line 7
      }
}
```

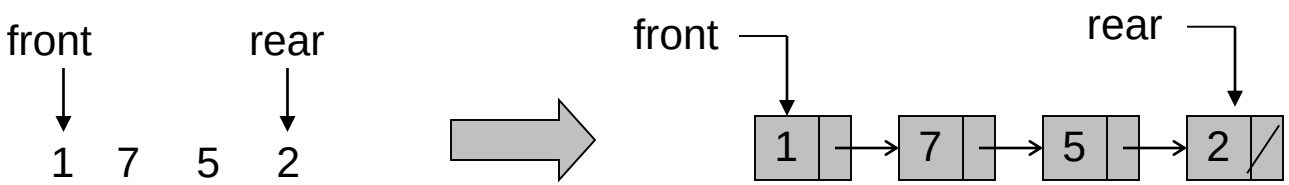


Queue.cpp

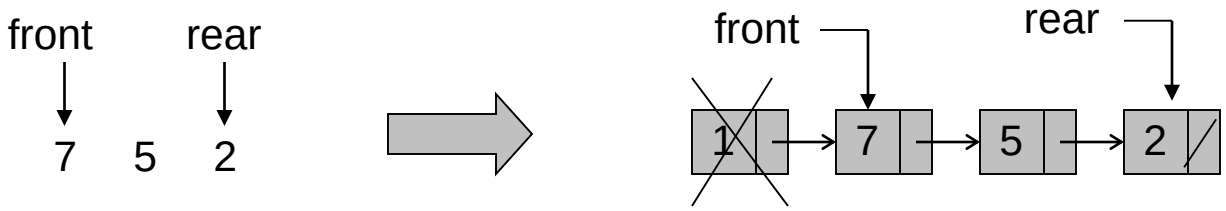
```
int Queue::dequeue()
{
    int x = front->get();
    Node* p = front;
    front = front->getNext();
    delete p;
    return x;
}
```

dequeue();

- Using linked List:



dequeue()



Queue.cpp

```
int Queue::front()
{
    return front->get();
}


int Queue::isEmpty()
{
    return ( front == nullptr );
}
```