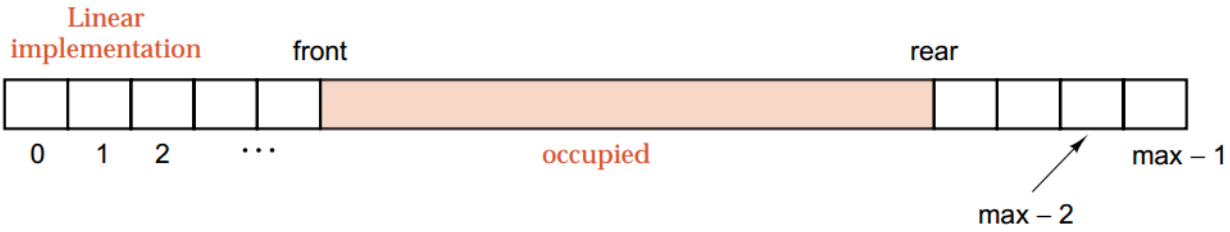
队列的说明

队列是一种基本的数据结构。我们平时排队等候服务的时候，是先到的人先接受服务。对于队列这种数据结构，实现的就是一种先进先出（first-int, first-out, FIFO）的策略。
改变队列中元素的操作方法只有两个——push与pop。push是把元素从队尾插入，pop是把元素从队头删除。

数组实现

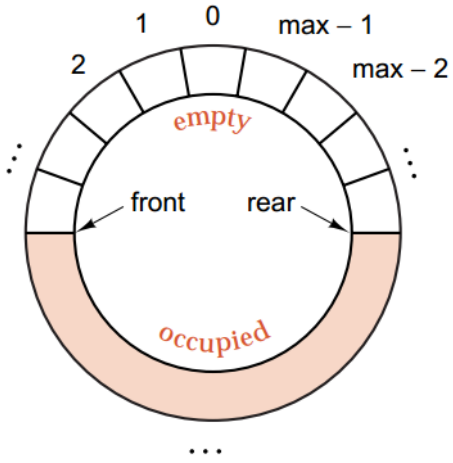
队列的数组实现有两种形式，一种是线性实现，另一种是循环实现。

线性实现有一定的局限性，当rear抵达max-1的时候就不能再让元素入队了。此时front前面的空间将被浪费，因为没有被使用。



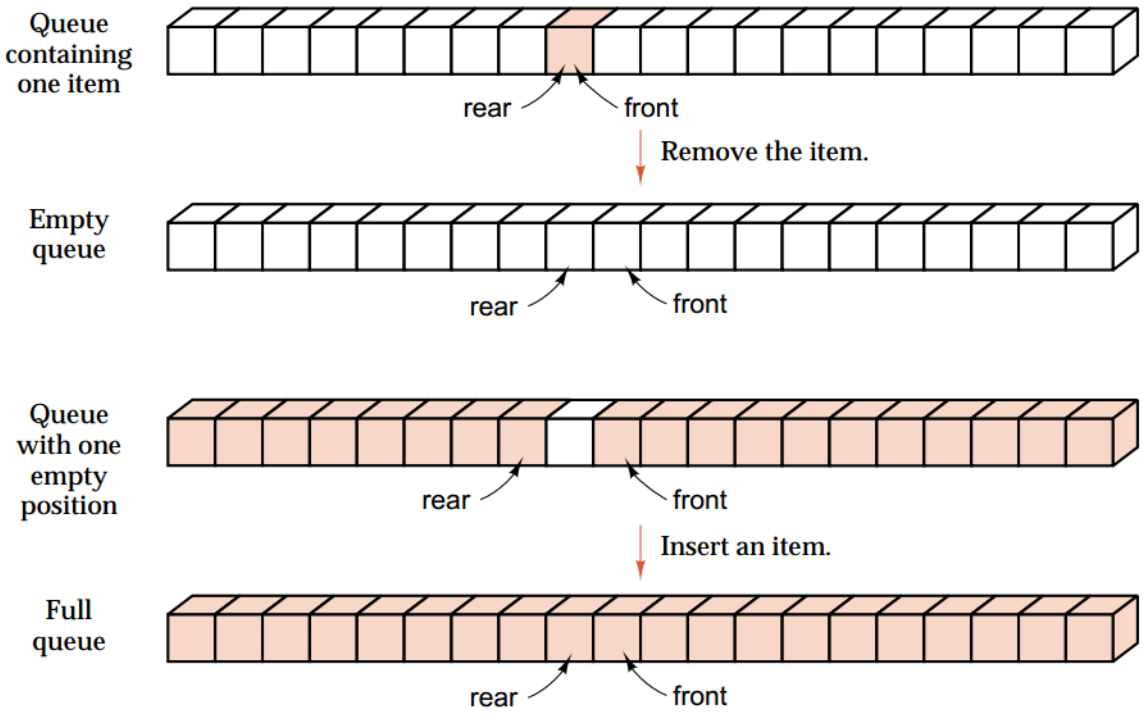
图片来自《Data Structures and Program Design In C++》

着重说说循环实现。
循环实现就是把一个数组看成一个循环的圆，当rear或front抵达max - 1的时候，再前进，将回到0处。这样一来，就能对数组的空间有充分的利用。



图片来自《Data Structures and Program Design In C++》

数组实现队列最大的问题在于，如何判断其边界，下面给出边界情况



图片来自《Data Structures and Program Design In C++》

在实现的时候，需要区分如何判断队列已满或者队列已空。

当rear的下一个位置是front的时候，则队列已满。

当rear和front的位置重叠，则队列已空。

这样就能避免图中出现的边界情况，难以判断满或空。

[cpp]view plaincopy

```cpp
template <class T>
class Queue {
private:
staticconstint maxqueue = 10;
  T entry[maxqueue + 1];  // 分配比maxqueue + 1的空间
// 实际上队列的可用空间还是maxque
int head;
int rear;
int queueSize;
bool full() const {
return (((rear + 1) % (maxqueue + 1)) == head)
            && (queueSize == maxqueue);
  }
public:
  Queue() : head(0), rear(0), queueSize(0) {
for (int i = 0; i < maxqueue + 1; i++)
      entry[i] = T();
  }
// 下面所以求余都是为了让rear和front的范围限定在[0, 10]
void push(const T &val) {
if (!full()) {
      entry[rear] = val;
      rear = (rear + 1) % (maxqueue + 1);
      ++queueSize;
    }
  }
void pop() {
if (!empty()) {
      head = (head + 1) % (maxqueue + 1);
      --queueSize;
    }
  }
  T front() const {
if (!empty())
return entry[head];
```

```
36.   }
37.   T back() const {
38. if (!empty())
39. // rear-1是队列最后一个元素的位置
40. // rear-1可能是-1，所以将其跳转到maxqueue位置
41. return entry[(rear - 1 + (maxqueue + 1)) % (maxqueue + 1)];
42.   }
43. bool empty() const {
44. return ((rear + 1) % (maxqueue + 1) == head) && (queueSize == 0);
45.   }
46. int size() const {
47. return queueSize;
48.   }
49. };
```

链表实现

链表实现利用两个结点，head记录链表头即队头，rear记录链表尾即队尾

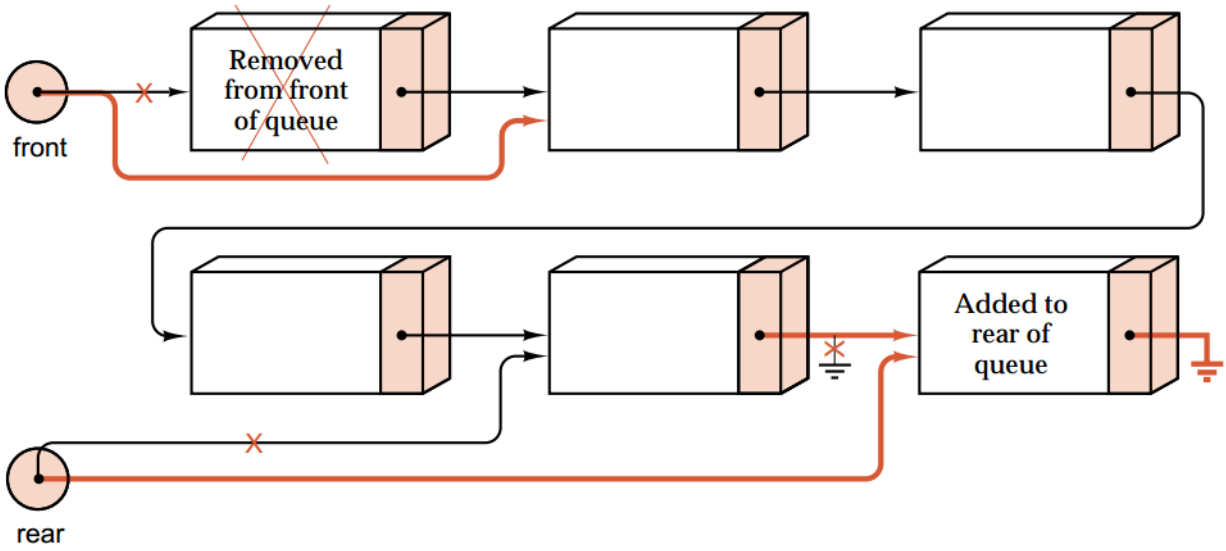


Figure 4.13. Operations on a linked queue

图片来自《Data Structures and Program Design In C++》

[cpp]view plaincopy

```
1. template <class T>
2. class Queue {
3. private:
4. struct Node {
5.     T data;
6.     Node *next;
7.   };
8. private:
9.   Node *head;
10.   Node *rear;
11. int queueSize;
12. public:
13.   Queue() : head(NULL), rear(NULL), queueSize(0) { }
14.   ~Queue() {
15. while (!empty())
16.       pop();
17.   }
18. void push(const T &val) {
19.     Node *new_node = new Node;
20.     new_node->data = val;
21.     new_node->next = NULL;
```

```
22. if (head == NULL && rear == NULL) {
23.       head = rear = new_node;
24.     } else {
25.       rear->next = new_node;
26.       rear = new_node;
27.     }
28.     ++queueSize;
29.   }
30. void pop() {
31. if (!empty()) {
32.       Node *temp = head;
33.       head = head->next;
34. delete temp;
35.       --queueSize;
36.     }
37.   }
38.   T front() const {
39. if (!empty())
40. return head->data;
41.   }
42.   T back() const {
43. if (!empty())
44. return rear->data;
45.   }
46. bool empty() const {
47. return queueSize == 0;
48.   }
49. int size() const {
50. return queueSize;
51.   }
52. };
```

参考书籍

《Data Structures and Program Design In C++》