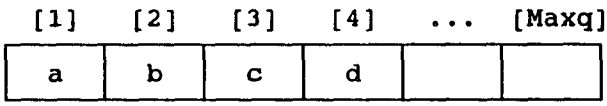
A BETTER QUEUE DESIGN

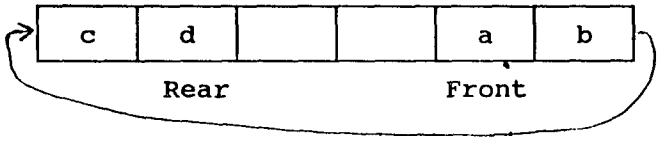
Charles L. Silver
Department of Computer Science
Southeastern Louisiana University
Hammond, LA 70402

Some Data Structures textbooks present the following scenario regarding queues: First we implement a queue as an array and make the Front of the queue coincide with the first position of the underlying array. We let the Rear of the queue be the position of the array containing the last element of the queue, like this:

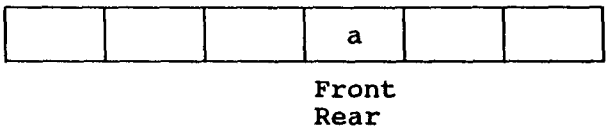


It is then then easy to add elements to the queue by first incrementing Rear by 1 and then storing the new element in that position in the array. But, it is not so easy to remove an element, according to the scenario. The reason for this is that fixing the Front of the queue in position 1 of the array requires us to shove everything down one whenever we remove the first element. This consumes processing time, especially if the queue is long or contains complex elements.

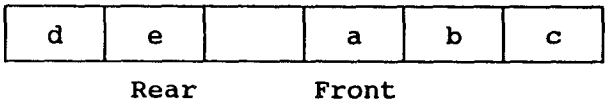
Another solution is to let the Front float toward the Rear of the queue as elements are removed. Then, to make use of the available space vacated by the removal of these elements, we create a "wrap-around" or "circular" queue where the Rear continues past the maximum size of the queue ("Maxq") to position 1, then 2, etc.:



This leads to a problem. The problem is to distinguish an empty queue from a full one. Suppose there is only one element in a circular queue and Front = Rear, as follows:



Then, when one element is removed, Front = Rear + 1. Now, suppose that the queue is one element shy of being full, like so:



Here, Front = Rear + 2. But then, when an element is added, Rear is incremented by 1 and Front = Rear + 1, the same as when the queue is empty. Thus, we have the undesirable result that when Rear + 1 = Front the queue is both empty and full.

The next step in the scenario is to modify the design so that a full queue and an empty one can be easily and clearly distinguished.

One such modification requires two changes. The first change is to consider the queue full when all but one of the cells are filled. Thus, the diagram above would represent a full queue. The second change is to consider the reserved space before the first element to be the Front. The ostensible reason for letting the Front of the queue be the space before the first element is the same reason as above: to distinguish a full queue from an empty one. This ends the scenario, which justifies the resulting queue design.

This final design is quite unintuitive. A queue is supposed to be like a line of shoppers at a check-out stand in a supermarket or a line of

people queuing up to buy tickets to a movie or a line of cars at a toll booth. These lines all must be thought of as beginning one position in front of their first elements if we are to satisfy the present queue design. Students encountering this design in their textbooks are clearly disturbed by its lack of intuitiveness, though they eventually accept it as being forced on them by the requirement that an empty queue be clearly distinguishable from a full one.

What is surprising, though, is that this design is not forced on us at all. Once an extra space has been added to the original queue design, the problem of distinguishing an empty queue from a full one is solved, and there are no other technical problems to resolve.

To show that this simple modification works, we initialize the queue so that Front = 1 and Rear = Maxq. This makes Rear + 1 (wrapping around) = Front, which is the case when the queue is empty. The most unintuitive feature of this design is the initial setting. But, we will see shortly that this initial setting makes the most sense.

When an element enters a queue, Rear should be incremented by 1 to indicate that the rear of a line moves back one element. Thus, starting out with an empty queue, where Front = 1 and Rear = Maxq, the entry of one element makes Front = 1 and Rear = 1 as well (Maxq + 1 = 1 when the queue wraps around). This is as it should be: when there is a single element in a line the front of the line and the rear of the line are the same, and both occupy the first position.[1] This shows, I think, that it makes most sense to initialize Front to 1 and Rear to the maximum length of the queue.

There is no problem distinguishing a full queue from an empty one, since 'Rear + 2 = Front' signifies a full queue. This is also intuitive, or at least not unintuitive, since the reserved space is in between Rear and Front, making Front two positions -- the space plus one position more -- past Rear.

I have taught circular queues using both queue designs, the one where Front is before the first position and the one where Front is the first position, and I can say that the latter design is accepted much more readily and grasped more easily than the former one.

In the other queue design, Front = Maxq and Rear = 1 when the first element enters the queue. This seems highly unnatural.

Engineering and computer experts offer conceptual background and hands-on, problem-centered advice...on video!

Now available through ACM — high-quality videotapes sponsored by your profession's corporate leaders...

☐ **DAN INGALLS**

Introduction to Object-Oriented Programming
Sponsored by Apple Computer
"An excellent introduction to object-oriented programming by one of the originators." — Stephen Omohundro, International Computer Science Institute
Discusses the principles of OOP, showing how this approach enhances software flexibility and reusability by supplanting conventional procedure-calls with the message-sending mechanism. **45 min.**

☐ **RON GRAHAM**

The Steiner Problem and NP-Completeness
Sponsored by AT&T Bell Laboratories
"Using some pretty mathematics, Graham illustrates how NP hard problems should be attacked." — Richard Karp, UC Berkeley
50 min.

☐ **SEYMOUR CRAY**

History, Supercomputing, and Gallium Arsenide
Sponsored by Cray Research, LLNL, LANL
Insights on the history and evolution of supercomputing, Cray-1 and Cray-2 development, and the new role of gallium arsenide in the upcoming Cray-3 and Cray-4. **73 min.**

☐ **DAVID PATTERSON**

RISC Design and SPARC Architecture
Sponsored by Sun Microsystems
"Excellent...the right way to discuss industry products." — Daniel Duchamp, Columbia University
Shows three SPARC boards running at three clock rates, each taking more power and board area. Covers historical perspective, business context, future directions.
62 min.

acm PRESS

Each tape $32.00 for ACM members, $35.00 for nonmembers, plus California sales tax where applicable and $5.00/tape shipping and handling. Videos are part of UVC's Distinguished Lecture Series, Vol. II, 1989/1990. VHS only. Others available — please inquire.

Order from ACM Press Publications Manager, 11 W. 42 Street, New York, NY 10036.

University Video Communications
UVC
Distinguished Lecture Series Volume II, 1989/1990