Further examples include the following:

- Calls to large companies are generally placed on a queue when all operators are busy.
- In large universities, where resources are limited, students must sign a waiting list if all computers are occupied. The student who has been at a computer the longest is forced off first, and the student who has been waiting the longest is the next user to be allowed on.

A whole branch of mathematics known as **queuing theory** deals with computing, probabilistically, how long users expect to wait on a line, how long the line gets, and other such questions. The answer depends on how frequently users arrive to the line and how long it takes to process a user once the user is served. Both of these parameters are given as probability distribution functions. In simple cases, an answer can be computed analytically. An example of an easy case would be a phone line with one operator. If the operator is busy, callers are placed on a waiting line (up to some maximum limit). This problem is important for businesses, because studies have shown that people are quick to hang up the phone.

If there are k operators, then this problem is much more difficult to solve. Problems that are difficult to solve analytically are often solved by a simulation. In our case, we would need to use a queue to perform the simulation. If k is large, we also need other data structures to do this efficiently. We shall see how to do this simulation in Chapter 6. We could then run the simulation for several values of k and choose the minimum k that gives a reasonable waiting time.

Additional uses for queues abound, and as with stacks, it is staggering that such a simple data structure can be so important.

Summary

This chapter describes the concept of ADTs and illustrates the concept with three of the most common abstract data types. The primary objective is to separate the implementation of the ADTs from their function. The program must know what the operations do, but it is actually better off not knowing how it is done.

Lists, stacks, and queues are perhaps the three fundamental data structures in all of computer science, and their use is documented through a host of examples. In particular, we saw how stacks are used to keep track of function calls and how recursion is actually implemented. This is important to understand, not just because it makes procedural languages possible, but because knowing how recursion is implemented removes a good deal of the mystery that surrounds its use. Although recursion is very powerful, it is not an entirely free operation; misuse and abuse of recursion can result in programs crashing.

Exercises

3.1 You are given a list, `L`, and another list, `P`, containing integers sorted in ascending order. The operation `printLots(L,P)` will print the elements in `L` that are in positions specified by `P`. For instance, if $P = 1, 3, 4, 6$, the elements in positions 1, 3, 4, and 6 in `L` are printed. Write the procedure `printLots(L,P)`. You may use only the public STL container operations. What is the running time of your procedure?