pointers.
will try to provide material to help future learning of them all and at least topic

Thank you. To students who responded to my request for topic suggestions.

Discrete event simulation: An application of priority queues.

Time step simulation

Stacks, Queues, Priority Queues

Mix-and-Match

CSI 310: Lecture 25
class SList {
    Item A[N];
} static const int N=5;
SList MySL;

Element-by-element dynamic allocation

class DList {
    struct Node{
        Item data;
        Node *link;
    }
    Node *Head;
} template <class Item>
DList<int> MyDL;
MyDL.insert(15);
MyDL.insert(72);

Static sized allocation of a List.

Element-by-element dynamic allocation
```cpp
struct Node
{
    Item data;
    Node *link;
};

template <class Item>
CACHED dynamic allocation
Node *array;
int capacity;
...
int used;
...

class CList
{
    CList<int> MyCL(5);
    MyCL.insert(15);
    MyCL.insert(72);
    MyCL.insert(15);
    MyCL.insert(72);
    MyCL.insert(15);
    MyCL.insert(72);

    class CList
    {
    public:
        CList();
        ~CList();
        void insert(Item data);
        void display();
    private:
        array;
        capacity;
        used;
    };

    template <class Item>
    CACHED dynamic allocation
    Node *link;

    class Item
    {
        int data;
    };

    struct Node
    {
        Item data;
        Node *link;
    };

    CList();
    ~CList();
    void display();
};
```
(Linked) List of Statically Sized Lists:
buffers would be a better data structure choice. It is illustrated next.

For such new applications, one-by-one dynamic allocation or dynamically sized

small packets occur in telnet/ssh protocol traffic.

Thus, a maximum sized buffer for a packet whose data size is only 1-2 bytes. Such

much larger, like several megabytes. So, it is very wasteful of memory to allocate

network hardware and protocols may have packet size limits that are

number generated by software and yet to be transmitted.

the predictable is number of packets received and yet to be processed, and the

Network cards send and receive packets from the network media whose sizes are

computer running a network interface.

The preceding slide illustrates a data structure like that in every modern
(Linked) List of Dynamically Sized Lists:

NULL

0
3

4
9

2
5

8
The programming languages you use to express these structures are likely to change over the decades. But the structures themselves probably will continue to be useful.

The data structures used in the common higher level languages of the time were not present in the common higher level languages and dynamically allocated language because features like pointers, structures, and data were programmed in assembly.

The data structures illustrated with the diagrams were used by programmers learning to write a particular library (like STL) is not a main goal; even though it is easy and practical when it fits your current application.

Learning to use a particular library (like STL) is not a main goal; even though it is easy and practical when it fits your current application.

1. After conceiving of data structures like those illustrated by the above diagrams, try implementing them by writing C++ class definitions (data structures definitions plus member functions).

2. Conceiving of data structures by “mix-and-match” combining of data structures taught in books and courses.

3. Figuring out all the advantages and disadvantages of making one choice ("choice B") of data structure compared to another choice ("choice A").

The structures taught in books and courses
the oldest item in stock (FIFO). This is taught in accounting courses.
now be based on the wholesale price paid last (LIFO) or the wholesale price paid for
inventory item typically varies with when it was bought. Should the profit from a sale
selling a unit from an inventory. The wholesale price the business had paid to buy one
several accounting practices used by businessespeople to define what profit results from
In other words, a FIFO (First-in First-out) store, (LIFO and FIFO are two of

In other words, a FIFO (First-in First-out) store.

queue: Sequence in which insertion (at the „front“) and removal (at the „rear“) occur.

queue: Sequence in which insertion (at the „front“) and removal (at the „rear“) occur.

stack: Sequence in which insertion and removal occur only at ONE END

University at Albany Computer Science Dept.
queue is filled.

When the buffer becomes empty, the library will detect this, and call the system function to read up to another block again. This function typically returns the number of characters actually read, so the library can calculate how much data needs to be read next.

Remove them when they are accessed by say `getc` one character at a time.

When the buffer becomes empty, the library will detect this and call the system function to read up to another block again. This function typically returns the number of characters actually read, so the library can calculate how much data needs to be read next.

The `getc()` member function of the `istream` class is used to read characters from an input stream. However, except for hand-typed input from a terminal, it is more efficient for the process to call a system function to read (disk) the data in blocks of 256 or more characters. The `istream` library uses a buffer (i.e., a memory region) to hold characters when they are read in blocks. When the buffer is empty, the system function is called to read a block of characters, and the characters are then stored in the buffer. The `getc()` member function of the `istream` class removes and returns the earliest character in the buffer.

The most prevalent use of queues in computers worldwide is for buffer.
reduction operator % is useful here.

There are more or less elegant ways to program this; the C++ modular
array length from it to make it "wrap around" to the beginning of the array.
If either position advanced beyond the end of the array, code will subtract the
they are equal, the queue is empty.
variables or pointer variables indicate the rear and the front of the queue. If
manage an array as a circular buffer. Two position indicators (subscript
A new implementation idea for a queue (described in detail in the text) is to
implemented very much like those we covered for stacks.
dynamically allocated arrays. The member functions of queue classes are
implemented several different ways: with linked lists, static sized arrays, and
Like stacks and other sequence or list type containers, queues can be
The details are given in the chapter on tree applications. Items in a heap-ordered balanced binary tree; and NOF store them fully sorted. In other words, if the elements were sorted in decreasing order of priority, the greatest, among all the elements currently in the container. The removal operation will remove an element whose priority is a numerical value (or other kind of value from a linearly ordered set) called its priority. A priority queue is a container where every inserted element must be given with...
Computer scientists love to reduce one data structure problem to another. The item will always be the earliest inserted item that was not yet removed. Then, the top priority negative of the current time (or count of items). Then, the top priority item will always be the current time (or count of items). Then, the top priority item will always be the current time (or count of items). Then, the top priority item will always be the current time (or count of items).

To implement a queue, insert each item with its priority value given by the

Either a stack or a queue can be implemented using a priority queue.
accumulate some elements.

which the number of arrivals exceeds the number served, the queue will

that the rate the "customers" are served. If there is a period of time during

We mentioned that the length of a queue would be zero if the arrival rate is less

a for-loop to simulate the second by second passage of time.

simulation. One variable current-second is incremented by 1 each time through

We covered the "car wash" simulation from Chapter 8. It is a time-step type of
do 0. insert(E, -T);
with FUTURE time i, and
create new events E

......

Event::simulate(pG)

................

{ event::simulate(MYpG);
  time = event::time;
  in the priority queue
  // in the earliest event
  // in the earliest event
  while (event = MYpG.top())

Discrete event simulation
(\textit{el-cheapo}, standard, and \textit{super}) which take different amounts of time. So several cars can be washed at the same time, and qualities of washes implement more complex car wash simulations. These might be multiple "stalts" with this priority queue driven discrete event simulation pattern, we can easily prioritize.

Events are stored in the priority queue, so the earliest events have the highest priority in the future. Those new events to be scheduled at various times in the future. When an event is selected and then simulated, the simulation may cause one or occur in the future. The priority queue selects the next event scheduled to occur. Discrete event simulation relies on a priority queue to hold the events that will simulate. Such simulations are often done in social or scientific research.

Discrete event simulation is more efficient and more flexible than time step.