The University
Of Michigan

EECS402 Lecture 19

Andrew M. Morgan

Savitch N/A
Randomness In Programming
Simulations

Random Numbers

- Random numbers are often very important in programming
  - Suppose you are writing a program to play the game of roulette
    - The numbers that come up on the roulette wheel are random
    - Writing a program such that the numbers that come up the same every time doesn't make for a very interesting roulette game
  - Suppose you are writing a simulation of a customer walking through the airport
    - Statistics can be obtained about this type of simulation
    - "A customer enters the airport on average once every 4 seconds", or "A customer spends, on average 8 minutes in a bookstore"
    - Your simulation might result in generating a new customer after a random time interval (given a uniform distribution around 4 seconds)

Computers and Random Numbers

- Computers can not generate truly "random" numbers
- Instead, pseudo-random numbers are used
  - Pseudo-random numbers are generated using a mathematical formula

Consider this formula: next = int((prev / 50) + prev * 17)

With a start value of 4, the numbers generated would be:
4, 68, 1157, 19692, 335157, ...

This list is not very "random" looking - it is *always increasing* at a fast rate

A Better Pseudo-Random Formula?

- The Midsquare method is another method of generating random numbers.
- To generate numbers of X digits, square the previous number and use the middle X digits

Start with the number 384 (for three digit pseudo-random #s)

$384^2 = 01\mathbf{474}56 \Rightarrow 474$	$069^2 = 00\mathbf{047}61 \Rightarrow 047$
$474^2 = 02\mathbf{246}76 \Rightarrow 246$	$047^2 = 00\mathbf{022}09 \Rightarrow 022$
$246^2 = 00\mathbf{605}16 \Rightarrow 605$	$022^2 = 00\mathbf{004}84 \Rightarrow 004$
$605^2 = 03\mathbf{660}25 \Rightarrow 660$	$004^2 = 00\mathbf{000}16 \Rightarrow 000$
$660^2 = 04\mathbf{356}00 \Rightarrow 356$	$000^2 = 00\mathbf{000}00 \Rightarrow 000$
$356^2 = 01\mathbf{267}36 \Rightarrow 267$	$000^2 = 00\mathbf{000}00 \Rightarrow 000$
$267^2 = 00\mathbf{712}89 \Rightarrow 712$	$000^2 = 00\mathbf{000}00 \Rightarrow 000$
$712^2 = 05\mathbf{069}44 \Rightarrow 069$	

This generator looks better at first. It isn't constantly increasing like the last one. However, it is *repeating*. Once it gets to the number 0, the number 0 is repeated over and over, which makes for a poor generator.

Another Generator

- The midsquare method on the previous page resulted in the value 0 being repeated after it is generated
- Some generators actually result in repeating patterns, which sometimes can be difficult to find
- The Linear Congruential Method uses the formula
  - xn+1 = (a xn + c) % m
  - If a==2, c==3, m==10, and x0 == 0, the sequence is:

$$\left.\begin{aligned} x_1 &= (2 \text{ X } 0 + 3) \bmod 10 = 3 \\ x_2 &= (2 \text{ X } 3 + 3) \bmod 10 = 9 \\ x_3 &= (2 \text{ X } 9 + 3) \bmod 10 = 1 \\ x_4 &= (2 \text{ X } 1 + 3) \bmod 10 = 5 \end{aligned}\right\}$$

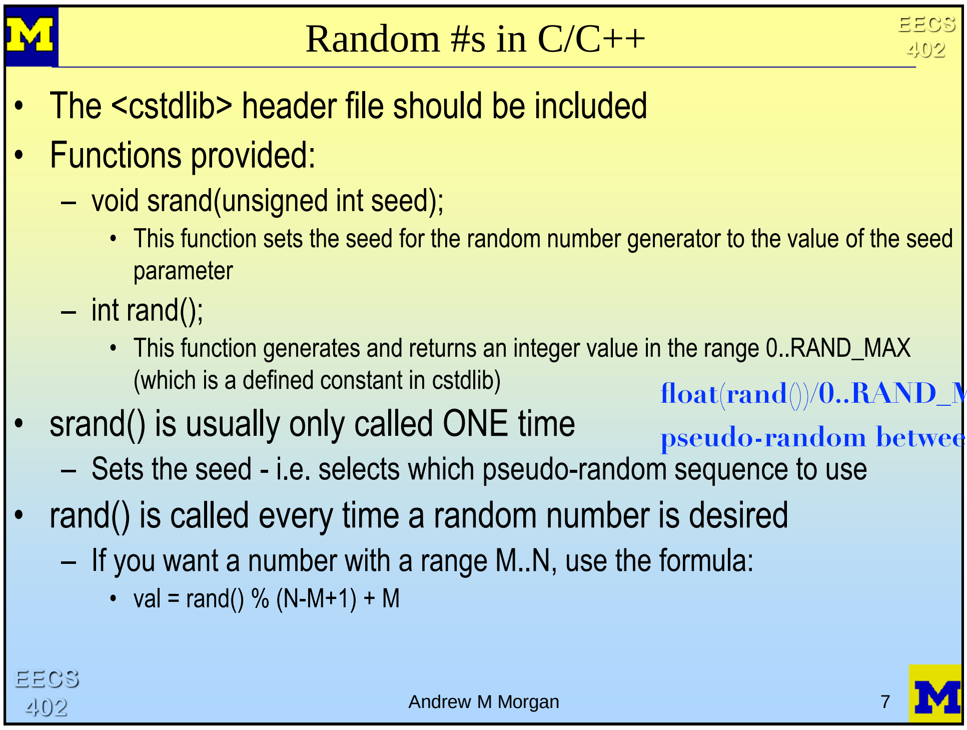
$$\left.\begin{aligned} x_5 &= (2 \text{ X } 5 + 3) \bmod 10 = 3 \\ x_6 &= (2 \text{ X } 3 + 3) \bmod 10 = 9 \\ x_7 &= (2 \text{ X } 9 + 3) \bmod 10 = 1 \\ x_8 &= (2 \text{ X } 1 + 3) \bmod 10 = 5 \end{aligned}\right\}$$

$$x_9 = (2 \text{ X } 5 + 3) \bmod 10 = 3$$

Repeating Pattern

An Advantage...

- The same sequence of pseudo-random numbers can be generated
- This is good, because it is difficult to debug programs that use truly random numbers
- By providing the same start-value (called a seed), the same sequence results, which means you can generate the same results time and time again
- When you are through debugging, you can use a different seed each time.
  - Often, this is done by using a portion of the system clock



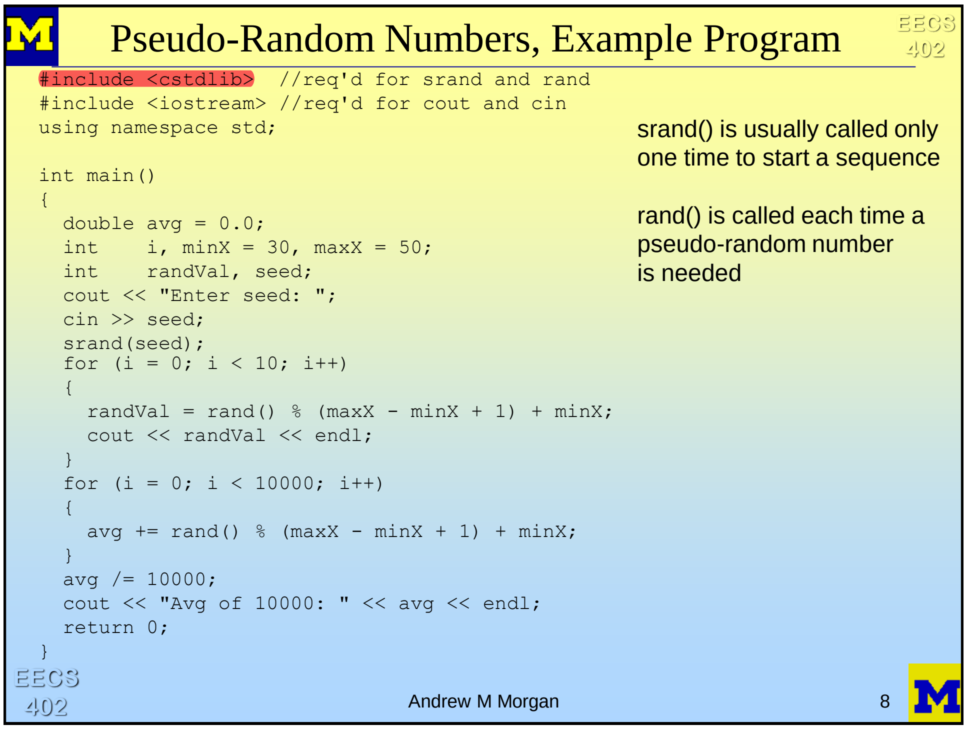
Random #s in C/C++

EECS 402

- The <cstdlib> header file should be included
- Functions provided:
  - void srand(unsigned int seed);
    - This function sets the seed for the random number generator to the value of the seed parameter
  - int rand();
    - This function generates and returns an integer value in the range 0..RAND_MAX (which is a defined constant in cstdlib)
- srand() is usually only called ONE time
  - Sets the seed - i.e. selects which pseudo-random sequence to use
- rand() is called every time a random number is desired
  - If you want a number with a range M..N, use the formula:
    - val = rand() % (N-M+1) + M

float(rand())/0..RAND_MAX to get a pseudo-random between 0 and 1

EECS 402 Andrew M Morgan 7



Pseudo-Random Numbers, Example Program

EECS 402

C standard library

```
#include <cstdlib>   //req'd for srand and rand
#include <iostream> //req'd for cout and cin
using namespace std;

int main()
{
  double avg = 0.0;
  int    i, minX = 30, maxX = 50;
  int    randVal, seed;
  cout << "Enter seed: ";
  cin >> seed;
  srand(seed);
  for (i = 0; i < 10; i++)
  {
    randVal = rand() % (maxX - minX + 1) + minX;
    cout << randVal << endl;
  }
  for (i = 0; i < 10000; i++)
  {
    avg += rand() % (maxX - minX + 1) + minX;
  }
  avg /= 10000;
  cout << "Avg of 10000: " << avg << endl;
  return 0;
}
```

srand() is usually called only one time to start a sequence

rand() is called each time a pseudo-random number is needed

EECS 402 Andrew M Morgan 8

Pseudo-Random Numbers, Example Output

```
[ 34 ] temp -: ./a.out       [ 35 ] temp -: ./a.out       [ 36 ] temp -: ./a.out
Enter seed: 12               Enter seed: 1652             Enter seed: 12
42                           38                           42
46                           32                           46
40                           43                           40
41                           43                           41
40                           36                           40
43                           48                           43
32                           34                           32
38                           48                           38
31                           31                           31
42                           49                           42
Avg of 10000: 39.9971        Avg of 10000: 40.0484        Avg of 10000: 39.9971
```

Note: Same seed = same sequence = same results

C++ Random # Generator

- The C++ random number generator DOES produce a repeating sequence of pseudo-random numbers
- The "period" of the generator is so large, though, that it is not usually a problem
  - The "period" of a random number generator is how large the pattern that repeats is
  - One version of C++'s random number generator has a period of 2^{32}
  - rand() will re-start the same repeating sequence after generating about *4.3 billion* random numbers
  - This is large enough for most applications requiring random number generation

C++ Generator Period Demo

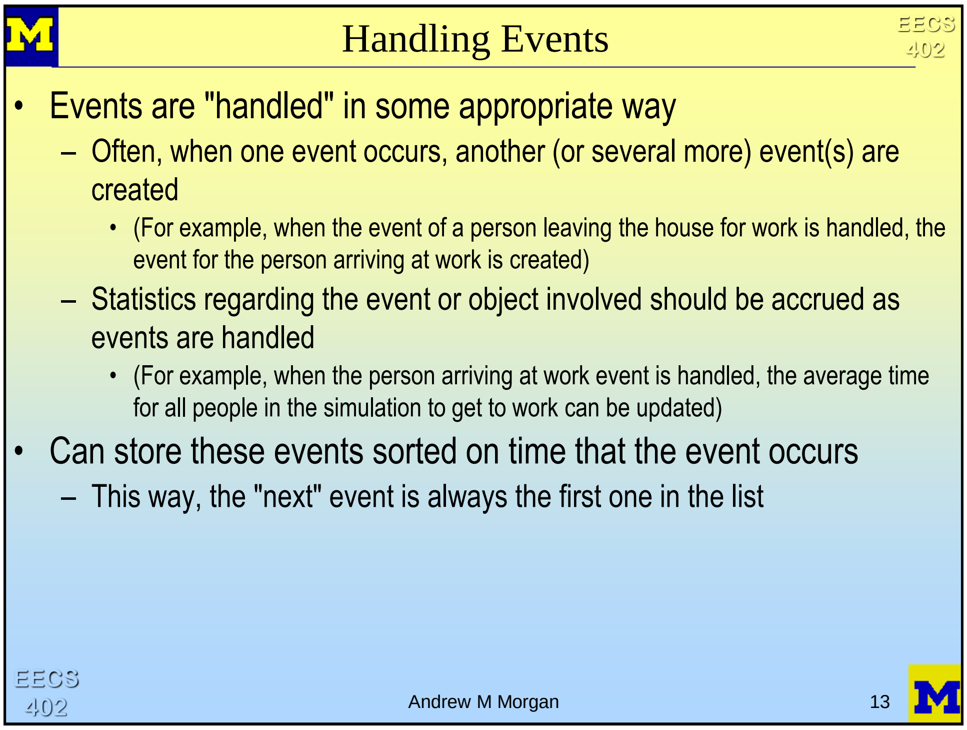
```
//include iostream, cstdlib, and cmath
int main()
{
  long int stopX;
  srand(100);
  stopX = (long int)pow(2, 32) - 1;
  long int li;

  cout << "RANDOM #" << rand() << endl;
  cout << "RANDOM #" << rand() << endl;
  for (li = 0; li < stopX; li++)
    rand();
  cout << "RANDOM #" << rand() << endl;
  cout << "RANDOM #" << rand() << endl;
  for (li = 0; li < stopX; li++)
    rand();
  cout << "RANDOM #" << rand() << endl;
  cout << "RANDOM #" << rand() << endl;
  return 0;
}
```

```
RANDOM #12662
RANDOM #23392
RANDOM #12662
RANDOM #23392
RANDOM #12662
RANDOM #23392
```

While the pattern DOES repeat, there are about 4.3 *billion* "random" numbers generated in the pattern. (This program took about 29 minutes of time to run)

Event-Driven Simulation

- In an event-driven simulation, time advances in an irregular way
  - If nothing happens (no events occur) for an hour, there is no reason to advance time one minute (or one second) at a time
  - Can "fast-forward" to the time when something important (an "event") occurs
- Consider a simulation of an airport terminal again
  - Events:
    - Traveler arrives
    - Traveler arrives at ticket agent line
    - Traveler finishes being served by ticket agent
    - Traveler arrives at gate
    - Traveler boards aircraft
    - etc...

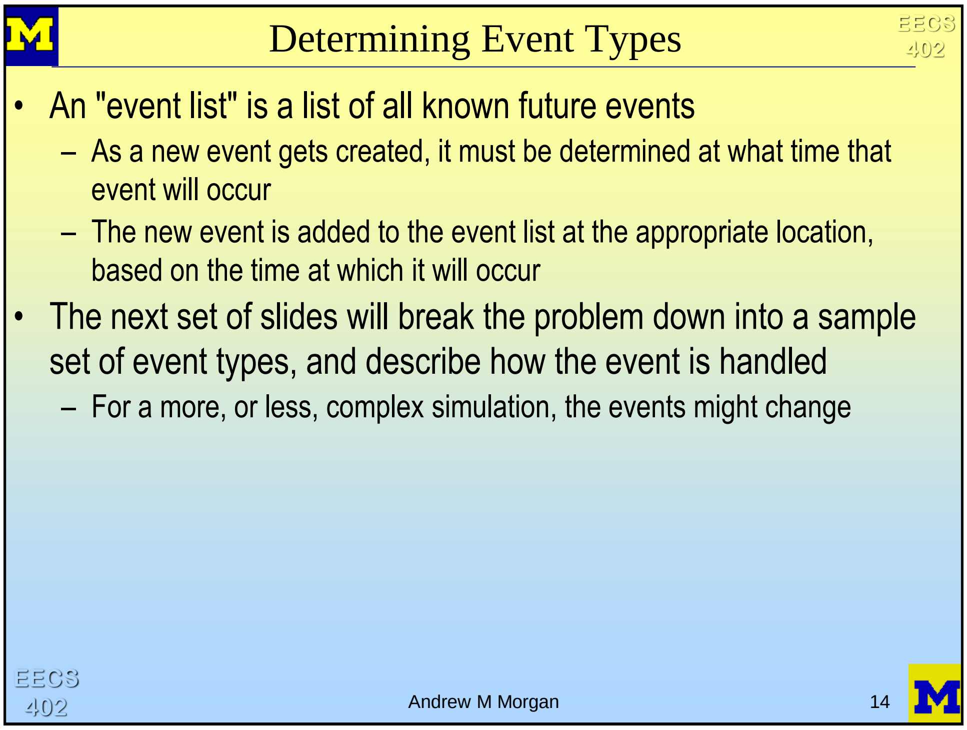


Handling Events

EECS 402

- Events are "handled" in some appropriate way
  - Often, when one event occurs, another (or several more) event(s) are created
    - (For example, when the event of a person leaving the house for work is handled, the event for the person arriving at work is created)
  - Statistics regarding the event or object involved should be accrued as events are handled
    - (For example, when the person arriving at work event is handled, the average time for all people in the simulation to get to work can be updated)
- Can store these events sorted on time that the event occurs
  - This way, the "next" event is always the first one in the list

EECS 402 Andrew M Morgan 13



Determining Event Types

EECS 402

- An "event list" is a list of all known future events
  - As a new event gets created, it must be determined at what time that event will occur
  - The new event is added to the event list at the appropriate location, based on the time at which it will occur
- The next set of slides will break the problem down into a sample set of event types, and describe how the event is handled
  - For a more, or less, complex simulation, the events might change

EECS 402 Andrew M Morgan 14

Customer Arrival At Airport Event

Event Description	Customer arrives at airport
Event Name	AIR_ARV
Handle Actions	1. Determine when next customer will arrive – Must do initial research at airport to record customer arrival times. Must determine arrival time distribution from recorded times, as well as distribution statistics (i.e. mean, standard deviation). Consider separating time of day into multiple segments with different distributions and/or statistics, to support busy time (rush hour), slow times, etc.) 2. Add a new AIR_ARV event for the next customer arrival to the event list 3. Determine customer's initial destination (ticket agent, restroom, self-serve kiosk, security checkpoint) – Must do initial research at airport to determine what percentage of newly arriving customers make each choice their first destination. 4. Determine time customer will arrive at chosen destination – Must do initial research to determine distribution type and statistics of how fast airport customers walk, as well as how the existing or planned airport will be laid out (distance from entrance to each possible destination, etc) 5. Add a new LOC_ARV event for the current customer's arrival at their initial destination

Customer Arrival At Location Event

Event Description	Customer arrives at a destination within the airport
Event Name	LOC_ARV
Handle Actions FIFO Queue	1. Add the current location to the customer's list of visited locations 2. If there is no line to wait in at the location: • Determine when the customer will be finished at the location – Must do initial research to determine distribution types and statistics on service times at the ticket agent counter, amount of time spent in restroom, time spent at kiosk, etc.) • Add a new LOC_DEP event to the event list to indicate when the customer will depart that location 3. Otherwise if there was a line to wait in at the location: • Insert customer at the end of the line waiting at the location – Note: no times are computed, because time at which customer will begin service at the location depends on the time required by the customers in front of him/her in the line

Customer Departs Location Event

Event Description	Customer finishes at location, and departs
Event Name	LOC_DEP
Handle Actions	1. Determine the customer's next destination within the airport, and the time at which the customer will arrive at that destination – Based on research of location visit patterns 2. Add a new LOC_ARV event for this customer's arrival at next destination 3. Check line at location being departed to determine if this customer's spot at this location can be taken by a waiting customer • If so: • Determine when the waiting customer will be finished at the location – Must do initial research to determine distribution types and statistics on service times at the ticket agent counter, amount of time spent in restroom, time spent at kiosk, etc.) • Add a new LOC_DEP event for when the customer that was waiting will depart this location

Customer Arrives At Gate Event

Event Description	Customer arrives at aircraft gate
Event Name	GATE_ARV
Handle Actions	1. None At this point, the customer has reached the gate, and the simulation no longer needs to create new events for this customer, as the action of boarding the plane is not a concern of this simulation. Note: This simulation may be combined with another simulation of air traffic from airport to airport in order to determine such information about how a weather delay in Atlanta will cause airport terminal congestion of customer's in Detroit – this would take into account time customers spend waiting for their plane to arrive, overbook situations requiring customers to be bumped to different flights, etc.

Sample Event-Driven Simulation

- Initial event list: AIR_ARV(cust 1, time 0)
- Time: 0, Handle first element in list, Traveler #1 arrives at airport
- Determine next traveler #2 arrival time (7), add to evt list
- Dest: Ticket Agent - Determine when traveler #1 arrives at Ticket Agent (4), add to evt list
- Event List: LOC_ARV(cust 1, time 4, loc TA), AIR_ARV(cust 2, time 7)
- Time: 4, Handle first element in list, Traveler #1 arrives at TA
- No line - Determine when traveler #1 will be done at TA (9), add to evt list
- Event List: AIR_ARV(cust 2, time 7), LOC_DEP(cust 1, time 9, loc TA)
- Time: 7, Handle first element in list, Traveler #2 arrives
- Determine next traveler #3 arrival time (12), add to evt list
- Dest: kiosk - Determine when traveler #2 arrives at kiosk (10), add to evt list
- Event List: LOC_DEP(cust 1, time 9, loc TA), LOC_ARV(cust 2, time 10, loc KIOSK), AIR_ARV(cust 3, time 12)
- Time: 9, Handle first element in list, Traveler #1 departs TA
- etc...

Notes On Sample Event-Driven Simulation

- Note the way time advanced: 0, 4, 7, 9
- Time advances irregularly
  - At times 1, 2, 3, 5, 6, and 8, nothing is happening in our airport
  - We can increase efficiency of the simulation by just skipping over those times
  - Nothing is missed because of this, since no events are occurring
- As one event is handled, other events are generated
- By sorting on event time, can always take first event
- Note: Events are generated on-the-fly
  - Dynamic allocation is used to create new events
  - Events are inserted into a sorted list
- Linked structure is used, rather than contiguous memory, since events may be inserted anywhere in the list

Time-Driven Simulation

- In a time-driven simulation, the simulation moves forward using constant time updates
  - For example, the state of the simulation is updated every minute
  - This does ***not*** imply a "real-time" simulation
  - A 24 hour time-driven simulation might only take 5 minutes to complete
  - A minute of simulation time can pass by updating all objects within the simulation, depending on their speed (if moving), etc.
- Example:
  - At time = 0, traveler i#1 is at (x,y) location (4, 2) in the airport, and is walking with a speed of (1,2) per minute
  - A minute of simulation time can pass by updating traveler #1's position to be (5, 4) at time = 1, then to (6, 6) at time = 2, etc.
  - In addition to locations, new customer's must arrive at correct times, etc.

What Type Of Simulation To Use?

- Event-driven simulations
  - Use when simulation can be broken down into events easily
  - Use when there is no good reason to use a time-driven simulation
- Time-driven simulations
  - Use when looking for path collisions, etc
    - For example, consider a national airline flight path simulation: Changing flight paths could have negative impacts (i.e. more mid-air collisions). An event driven simulation might be useful in analyzing how aircraft get backed up on the runway, etc, but would ***not*** be useful in analyzing how the change affects the number of mid-air collisions (can't define future events for mid0air collisions), whereas a time-drive simulation would move aircraft toward destination one time-unit at a time, and the space between aircraft could be analyzed at each time step
  - Use when attaching simulation to an animation
    - Event driven simulations would look odd if animated, due to variable time updates
    - Time-drive simulations look better since time moves in constant updates

But Why Randomness?

- Events do not happen in set intervals
  - Travelers do not show up every 4 minutes exactly
    - Sometimes a traveler comes in 30 seconds after the previous one
    - Sometimes, there are 15 minutes between arrivals
- Simulations are used to model the real world, so event generation must model actual events
  - Using exact intervals could result in unexpected special cases
  - If travelers arrive every 4 minutes, and ticket agents take 3.5 minutes to serve a traveler, no line will ever form at the ticket agent
  - Realistically, though, several customers might come in during a 4 minute period, resulting in a line forming at the ticket agent

Common Simulation Statistics

- Simulations are written to see "what would happen if…"
- Statistics are kept as the simulation runs
- Some common statistics are:
  - Average customer wait time
  - Maximum customer wait time
    - What was the longest time any customer had to wait in line?
  - Average server utilization
    - What percentage of the time were servers busy helping customers?
  - Minimum server utilization
    - Was there a server whose services were not needed?
  - Average line length
  - Maximum line length
  - Maximum number of customers in store at any given time

Usages Of Simulations

- Simulations are often used to determine "what would happen if..." when it isn't practical to just try it
- Experience: "Free-flight" system
  - Airlines want to file their own flight plans, resulting in more direct flight pattern, saving fuel, etc
  - The FAA doesn't want to say "Let's try it and see if it is chaotic or not"
  - A simulation is built of the national airspace system, using real flight data, etc
  - Simulation is run using current FAA regulations, and another is run using "free-flight" rules
  - Comparisons are done on results to determine whether "free-flight" is a safe, and the amount of savings that will result
  - Once determined safe, FAA regulations are set to allow free-flight

Usages Of Simulations, p.2

- Consider a traffic light system being installed at an intersection
- How long should the light stay green in each direction?
  - Statistics can be obtained about traffic flow on the two streets
  - Simulation can be written to simulate traffic on the roads, given specific traffic light parameters
  - Green light duration can be changed and simulation executed over and over
  - Average car wait times computed for each light configuration
  - Light is finally installed, and is set to stay green for duration that resulted in minimum average wait time

Usages Of Simulations, p.3

- Consider a new retail business being opened
- How many servers should be working during each shift?
  - Can keep customers happy by ensuring they don't have to wait in line and will always be helped by simply hiring 1000 servers for each shift
    - This will certainly result in many servers being completely unutilized, which means the new business pays them, but gets no utility from them
  - Could keep costs low by only hiring 1 server per shift
    - This will certainly infuriate customers, as they will often be required to wait for long periods of time to be served. Repeat business goes down, and profits are lost.
- A simulation could be written to determine best hiring policy
  - Simulate customers entering store, waiting for an available server, waiting in line to pay, etc. Vary the number of servers for several runs
  - Analyze statistics to determine average customer wait time, average server utilization, average customer satisfaction (if it can be characterized based on simulation parameters), etc.

Review: Event-Driven Simulation Flow

- Schedule the "seed" events (i.e. the events that are known when the simulation begins; first customer arriving, first virus introduced into population, etc.)
- Iterate until some stopping criteria (closing time, no more events to handle, etc.)
  - Get the next event that is scheduled to occur
  - Handle the event
    - Update the state of the simulation, statistics, etc., based on what to do when handling this type of event
    - Schedule "follow-up" events to occur in the future as a result of this event being handled
- Perform wrap-up operations (i.e. statistics calculations, result determinations, output, etc.)

- Obviously, the "handle the event" part is where the complexity is typically