CSIM: Part II

Instructor: Dr. Sunho Lim (Ph.D., Assistant Professor)

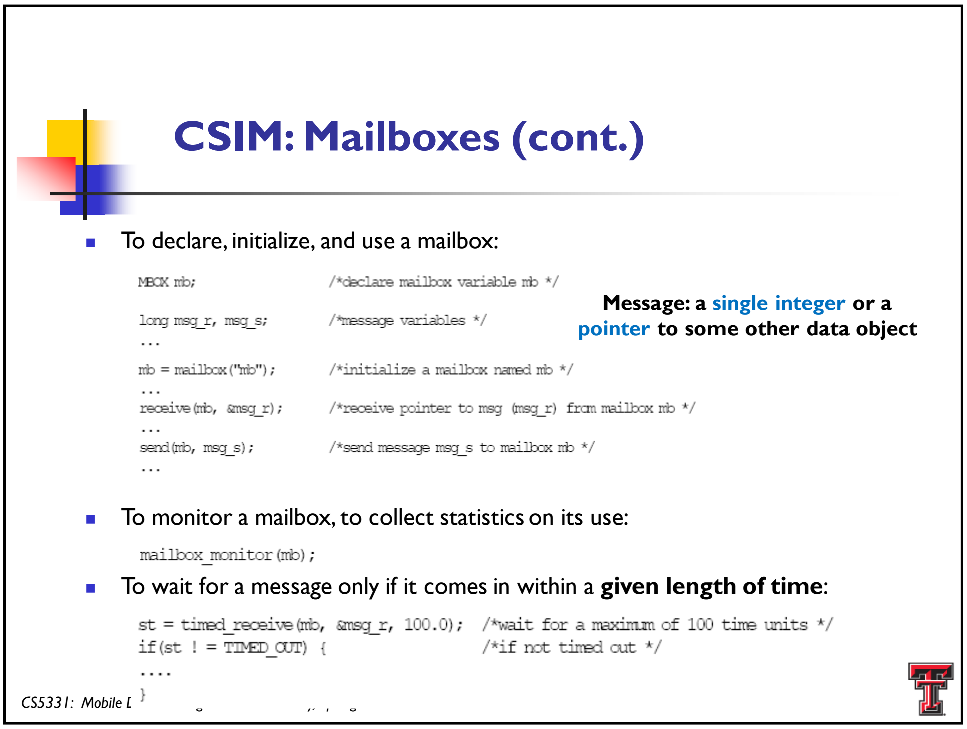
Lecture 05

sunho.lim@ttu.edu

Adapted partially from CSIM Document, http://www.mesquite.com/documentation

CSIM: Mailboxes

- Asynchronous exchange of data between CSIM processes
  - Any process may send a message to any mailbox
  - Any process may attempt to receive a message from any mailbox
  - **Message**: a single integer or a pointer to some other data object
- Two FIFO queues:
  - A queue of **unreceived messages**
  - A queue of **waiting processes**
  - At least one of the queues will be empty at any time
- When a process sends a message,
  - It is given to a waiting process (if one exists) or it is placed in the message queue.
- When a process attempts to receive a message,
  - It is either given a message from the message queue (if one exists) or it is added to the queue of waiting processes



CSIM: Mailboxes (cont.)

- To declare, initialize, and use a mailbox:

```
MBOX mb;                   /*declare mailbox variable mb */

long msg_r, msg_s;         /*message variables */
...
mb = mailbox("mb");        /*initialize a mailbox named mb */
...
receive(mb, &msg_r);       /*receive pointer to msg (msg_r) from mailbox mb */
...
send(mb, msg_s);           /*send message msg_s to mailbox mb */
...
```

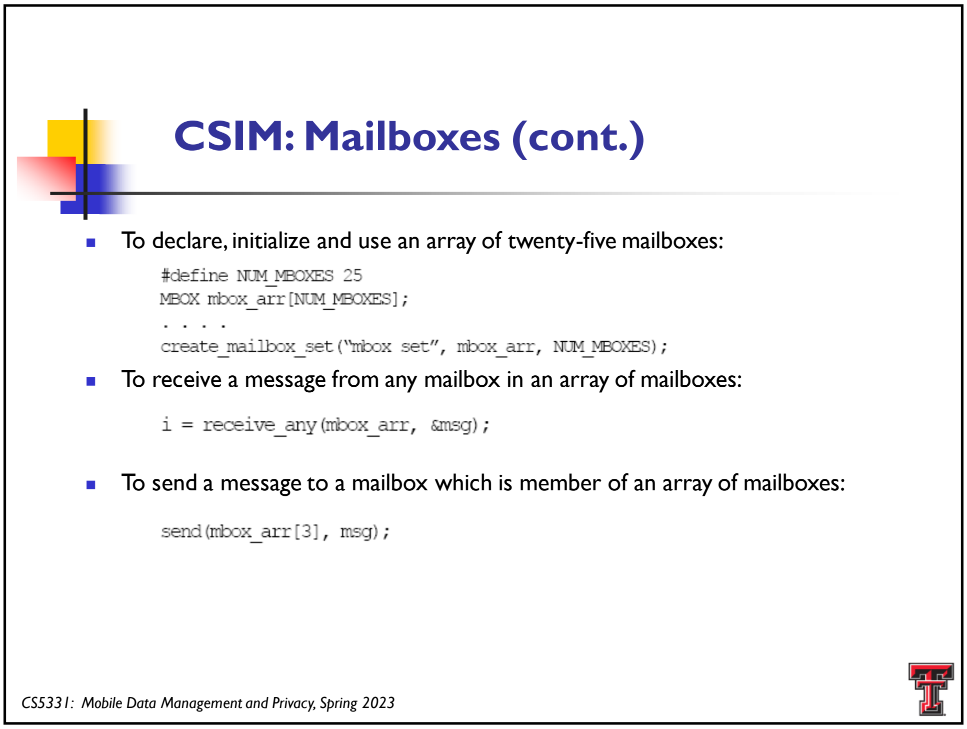
Message: a single integer or a pointer to some other data object

- To monitor a mailbox, to collect statistics on its use:

```
mailbox_monitor(mb);
```

- To wait for a message only if it comes in within a **given length of time**:

```
st = timed_receive(mb, &msg_r, 100.0);  /*wait for a maximum of 100 time units */
if(st ! = TIMED_OUT) {                  /*if not timed out */
....
}
```



CSIM: Mailboxes (cont.)

- To declare, initialize and use an array of twenty-five mailboxes:

```
#define NUM_MBOXES 25
MBOX mbox_arr[NUM_MBOXES];
. . . .
create_mailbox_set("mbox set", mbox_arr, NUM_MBOXES);
```

- To receive a message from any mailbox in an array of mailboxes:

```
i = receive_any(mbox_arr, &msg);
```

- To send a message to a mailbox which is member of an array of mailboxes:

```
send(mbox_arr[3], msg);
```

CSIM: Mailboxes (cont.)

- To receive a message from any mailbox in an array of mailboxes within a specified interval of time:

```
st = timed_receive_any(mbox_arr, &msg, 1.0);
if(st != TIMED_OUT) {
                        // process message
} else {
                        // deal with time out
}
```

- To send a message and wait until the message is received:

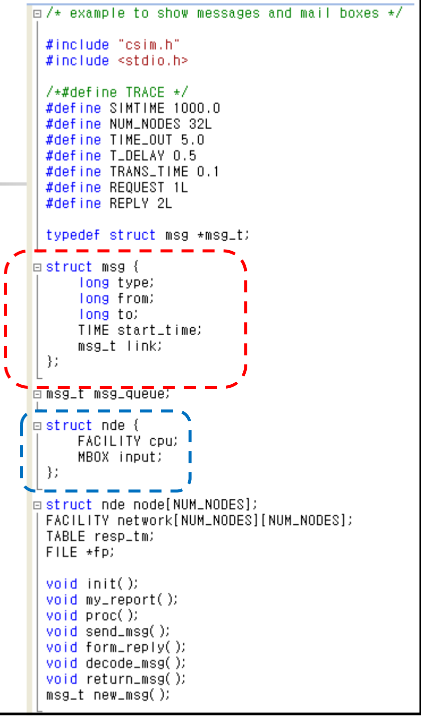
```
synchronous_send(mb, msg);
```

- To send a message and wait until the message is received within a specified interval of time:

```
st = timed_synchornous_send(mb, msg, 1.0);
if(st    != TIMED_OUT) {
                                // message received OK
} else {
                                // message not received
}
```

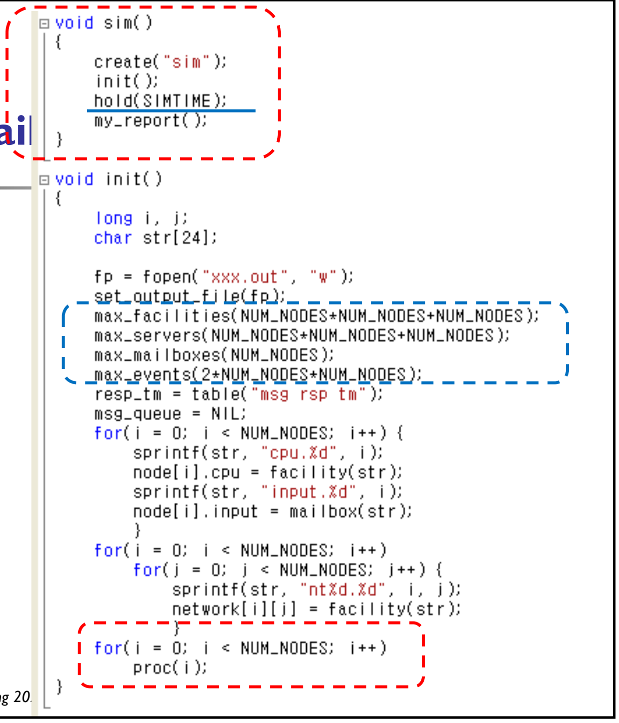
CSIM: Mailboxes (cont.)

- For example,



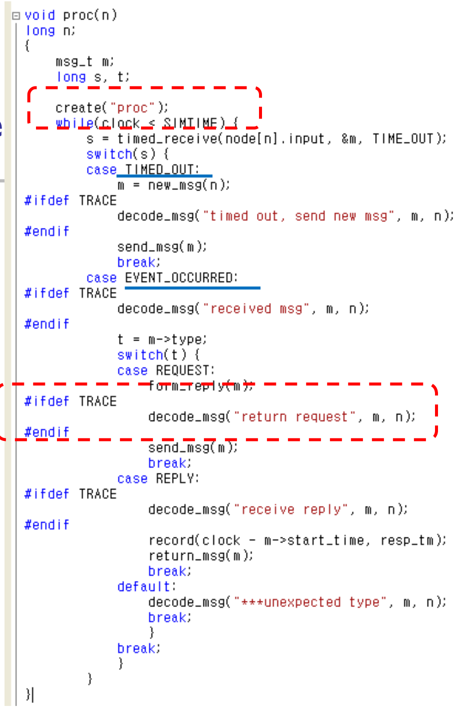
```
/* example to show messages and mail boxes */

#include "csim.h"
#include <stdio.h>

/*#define TRACE */
#define SIMTIME 1000.0
#define NUM_NODES 32L
#define TIME_OUT 5.0
#define T_DELAY 0.5
#define TRANS_TIME 0.1
#define REQUEST 1L
#define REPLY 2L

typedef struct msg *msg_t;

struct msg {
    long type;
    long from;
    long to;
    TIME start_time;
    msg_t link;
};

msg_t msg_queue;

struct nde {
    FACILITY cpu;
    MBOX input;
};

struct nde node[NUM_NODES];
FACILITY network[NUM_NODES][NUM_NODES];
TABLE resp_tm;
FILE *fp;

void init();
void my_report();
void proc();
void send_msg();
void form_reply();
void decode_msg();
void return_msg();
msg_t new_msg();
```

CSIM: Mai



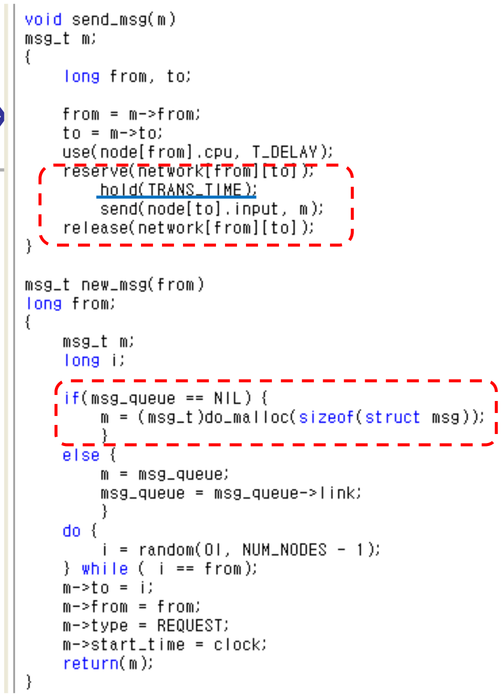
```
void sim()
{
    create("sim");
    init();
    hold(SIMTIME);
    my_report();
}

void init()
{
    long i, j;
    char str[24];

    fp = fopen("xxx.out", "w");
    set_output_file(fp);
    max_facilities(NUM_NODES*NUM_NODES+NUM_NODES);
    max_servers(NUM_NODES*NUM_NODES+NUM_NODES);
    max_mailboxes(NUM_NODES);
    max_events(2*NUM_NODES*NUM_NODES);
    resp_tm = table("msg rsp tm");
    msg_queue = NIL;
    for(i = 0; i < NUM_NODES; i++) {
        sprintf(str, "cpu.%d", i);
        node[i].cpu = facility(str);
        sprintf(str, "input.%d", i);
        node[i].input = mailbox(str);
        }
    for(i = 0; i < NUM_NODES; i++)
        for(j = 0; j < NUM_NODES; j++) {
            sprintf(str, "nt%d.%d", i, j);
            network[i][j] = facility(str);
            }
    for(i = 0; i < NUM_NODES; i++)
        proc(i);
}
```

CSIM: Mailboxe



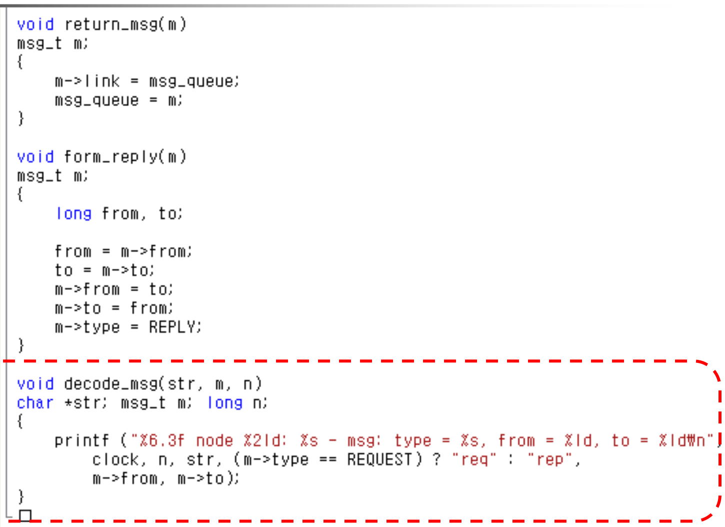
```
void proc(n)
long n;
{
    msg_t m;
    long s, t;

    create("proc");
    while(clock < SIMTIME) {
        s = timed_receive(node[n].input, &m, TIME_OUT);
        switch(s) {
        case TIMED_OUT:
            m = new_msg(n);
#ifdef TRACE
            decode_msg("timed out, send new msg", m, n);
#endif
            send_msg(m);
            break;
        case EVENT_OCCURRED:
#ifdef TRACE
            decode_msg("received msg", m, n);
#endif
            t = m->type;
            switch(t) {
            case REQUEST:
                form_reply(m);
#ifdef TRACE
                decode_msg("return request", m, n);
#endif
                send_msg(m);
                break;
            case REPLY:
#ifdef TRACE
                decode_msg("receive reply", m, n);
#endif
                record(clock - m->start_time, resp_tm);
                return_msg(m);
                break;
            default:
                decode_msg("***unexpected type", m, n);
                break;
                }
            break;
            }
        }
}
```

CSIM: Mailbox

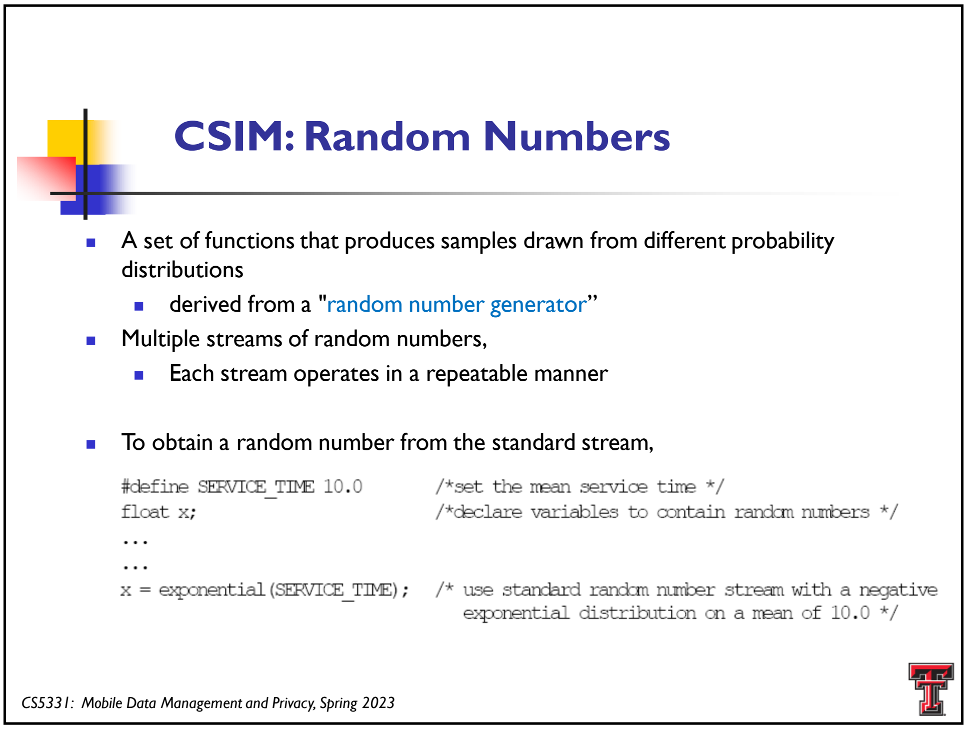


```
void send_msg(m)
msg_t m;
{
    long from, to;

    from = m->from;
    to = m->to;
    use(node[from].cpu, T_DELAY);
    reserve(network[from][to]);
        hold(TRANS_TIME);
        send(node[to].input, m);
    release(network[from][to]);
}

msg_t new_msg(from)
long from;
{
    msg_t m;
    long i;

    if(msg_queue == NIL) {
        m = (msg_t)do_malloc(sizeof(struct msg));
        }
    else {
        m = msg_queue;
        msg_queue = msg_queue->link;
        }
    do {
        i = random(0l, NUM_NODES - 1);
    } while ( i == from);
    m->to = i;
    m->from = from;
    m->type = REQUEST;
    m->start_time = clock;
    return(m);
}
```

CSIM: Mailboxes (cont.)



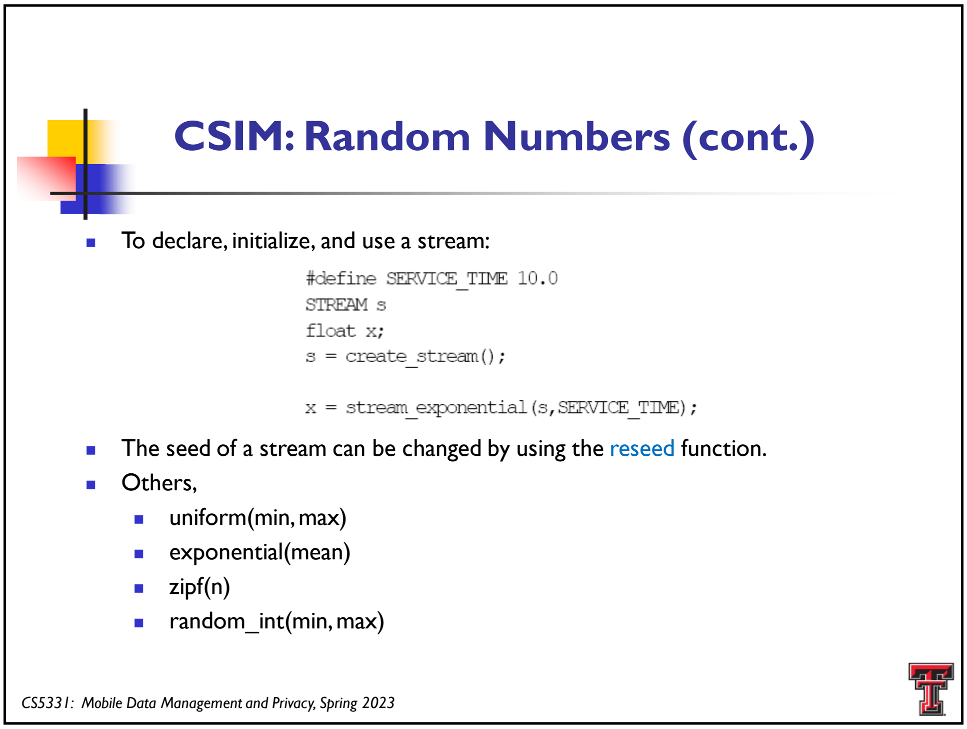
```
void return_msg(m)
msg_t m;
{
    m->link = msg_queue;
    msg_queue = m;
}

void form_reply(m)
msg_t m;
{
    long from, to;

    from = m->from;
    to = m->to;
    m->from = to;
    m->to = from;
    m->type = REPLY;
}

void decode_msg(str, m, n)
char *str; msg_t m; long n;
{
    printf ("%6.3f node %2ld: %s - msg: type = %s, from = %ld, to = %ld₩n",
        clock, n, str, (m->type == REQUEST) ? "req" : "rep",
        m->from, m->to);
}
```



CSIM: Random Numbers

- A set of functions that produces samples drawn from different probability distributions
  - derived from a "random number generator"
- Multiple streams of random numbers,
  - Each stream operates in a repeatable manner
- To obtain a random number from the standard stream,

```
#define SERVICE_TIME 10.0       /*set the mean service time */
float x;                        /*declare variables to contain random numbers */
...
...
x = exponential(SERVICE_TIME);  /* use standard random number stream with a negative
                                   exponential distribution on a mean of 10.0 */
```



CSIM: Random Numbers (cont.)

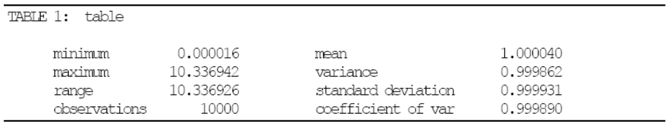
- To declare, initialize, and use a stream:

```
#define SERVICE_TIME 10.0
STREAM s
float x;
s = create_stream();

x = stream_exponential(s,SERVICE_TIME);
```

- The seed of a stream can be changed by using the reseed function.
- Others,
  - uniform(min, max)
  - exponential(mean)
  - zipf(n)
  - random_int(min, max)

CSIM: Tables and Qtables

- CSIM automatically collects some usage statistics
- Table
  - Collect floating point values and then gives a statistical summary

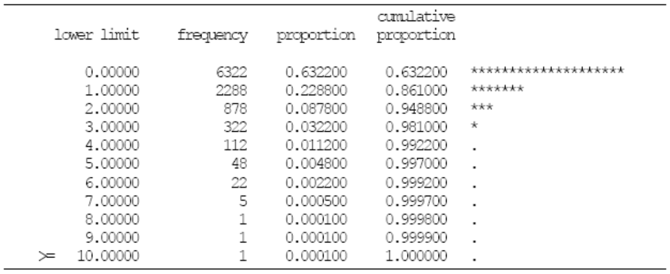


```
TABLE 1:  table

      minimum          0.000016          mean                     1.000040
      maximum         10.336942          variance                 0.999862
      range           10.336926          standard deviation       0.999931
      observations        10000          coefficient of var       0.999890
```

- A permanent table,
  - Not affected by requests to reset statistics or rerun the model, and can thus be used to gather data across multiple runs of a model

CSIM: Tables and Qtables

- Histogram
  - Obtain more detailed information about the recorded values
  - User-defined number of intervals and minimum and maximum values



```
                                              cumulative
      lower limit    frequency   proportion   proportion

          0.00000         6322     0.632200     0.632200   ********************
          1.00000         2288     0.228800     0.861000   *******
          2.00000          878     0.087800     0.948800   ***
          3.00000          322     0.032200     0.981000   *
          4.00000          112     0.011200     0.992200   .
          5.00000           48     0.004800     0.997000   .
          6.00000           22     0.002200     0.999200   .
          7.00000            5     0.000500     0.999700   .
          8.00000            1     0.000100     0.999800   .
          9.00000            1     0.000100     0.999900   .
    >=   10.00000            1     0.000100     1.000000   .
```

CSIM: Tables and Qtables

- Qtable
  - Track state changes (for example the number of processes in a queue)

```
QTABLE 1:  qtable

      initial      0     minimum     0     mean                  0.795029
      final        0     maximum     7     variance              0.802270
      entries  10000     range       7     standard deviation    0.895696
      exits    10000                       coeff of variation    1.126620
```

- To declare, initialize, and use a non-permanent table with a histogram:

```
TABLE tbl;                          /*declare table variable tbl */
...
tbl = table("tbl");                 /*initialize a table named tbl */
table_histogram(tbl,10,0.0,20.0); /*add a historgram to a table named tbl */
...
t = clock;                          /*get current time */
reserve(single_server);             /*reserve a single server facility */
     x = clock - t;                 /*calculate time spent on queue (delay  interval)*/
     tabulate(tbl, x);              /*record delay interval in table  */
...
```

CSIM: Tables and Qtables

- To declare, initialize, and use a non-permanent qtable with a histogram:

```
QTABLE qtble;                  /*declare qutable*/
...
qtbl=qtable("qtbl");           /* initialize qtable named qtbl */
qtable_histogram(qtbl,20,0,20);/* add histogram to qtable named qtbl */

note_entry(qtbl);              /*record entry onto queue for facility */
reserve(single_server);        /*reserve a single server facility */
     note_exit(qtbl);          /* record exit from queue for facility  */
     hold(exponential(2.5));
```