d e m o s   r e f e r e n c e   m a n u a l

by

graham birtwistle

computer science, university of bradford

acknowledgements

the demos system was implemented and tested on the leeds university centre for computer studies dec system 10 computer using the excellent simula compiler written by the swedish defence research establishment, stockholm. in addition, the demos text book and this manual were produced on the leeds machine by the author using the dec utilities sos and runoff.

the author is extremely grateful to the leeds university centre for computer studies for permission to use their computer, and would also like to record his appreciation of the centre's staff for their cheerful, able, and willing assistance over the years.

apologies

this implementation of demos follows the demos text]1[except on three points:

1) deadlock checking is not yet implemented.

2) the check when an entity becomes terminated to see that it does not hold any resources is not yet implemented. when it is, a tighter check will be made inside r.release(n)]ref(res)r;[.

n.b. 1) and 2) will be implemented together.

3) in the first implementation - for the demos text]1[- entities acquiring resources from reses or bins, or coopting entities from waitqs, were not entered into the appropriate queue if the request could be accommodated at once. it would seem to be more consistent if they are and so that implementation detail has been altered. as far as the examples in]1[are concerned, the practical effect is noticable only on the statistic qmax which may be as reported in]1[or 1 greater.

contents

3-15.

9.1 some primitive reporting aids

procedure clocktime 9-1; procedure box; procedure printreal 9-1; procedure edit 9-1.

chapter 1

introduction

this manual documents the latest version of demos. it is not a teaching text - for that see simula begin]2[and demos]1[; however demos]1[does not mention many of the features herein outlined.

demos has been implemented as a flat, monolithic context so that its external declaration is the same for every simula implementation:

external class demos

however it was coded in several logical layers, and a chapter is devoted to each.

layout of the manual

the rest of chapter 1 introduces it all by giving a demos model for a reasonably tough problem (another version of this problem is given in the ecsl manual]clementson, 3[).

chapter 2 covers the reporting aids class tab and class reportq. all demos facilities are prefixed by tab which, amongst other things, allows reportq membership. on generation, each facility object is entered into a special reportq reserved for its type. for example, all res objects are entered into resq, all waitq objects into waitqq, and so on. it is now very easy to write routines to report or reset each and every facility object created during program execution.

chapter 3 explains the data collection devices - count (incidences), tally (time independent data), histogram (tally plus a bar chart), accumulate (time dependent data), and regression (for linear regressions). each of these classes is prefixed by tab.

chapter 4 outlines the demos random number generators (taken from downham androberts]downham, 5[and its method of generating well spread seeds (due to ohlin]ohlin, 7[). well spread seeds are dealt with in 4.1; dist, the prefix to all distribution objects, in 4.2; distributions producing real results (constant, empirical, erlang, negexp, normal, and uniform) in 4.3; distributions producing integer results (poisson, randint) in 4.4; and distributions producing boolean results (draw) in 4.5. 4.6 outlines readdist, a global routine for creating distributions from descriptions supplied on inf.

chapter 5 documents class entity and its local scheduling routines. demos implements its own event list as a leftist priority tree. this is mainly for pedagogic reasons, but it doesn't degrade performance by much. should you wish to alter the event list strategy, e.g. by introducing priorities (this is given as an example on page 5-14), it is now much easier.

chapter 6 outlines the global scheduling routines hold, passivate; also time.

chapter 7 documents the queueing facilities. only demos entities may be queued; if you wish to queue other items, you will either have to write the routines yourself or use simset (demos doesn't). the types of queue implemented are: queue (usually for holding several coopted entities until they are required by their masters) in 7.1; waitq (master/slave synchronisation) in 7.2; and condq (waits until) in 7.3.

chapter 8 documents resource and its subclasses res (for the mutual exclusion synchronisation); and bin (for the producer/consumer synchronisation).

chapter 9 clears up most of the remaining odds and ends; the underlying reporting aids, tracing, error messages, report and reset, queue and event list snapping, and epsilon - the grain of time.

chapter 10 documents the demos prefix; the variables it contains and the prefix initialising and finalising actions.

the last chapter, chapter 11, explains how to redirect the standard input file inf and output file outf.

finally, there are the references; three appendices giving an indented listing of the demos source (appendix a), a set of 21 sample programs (appendix b); their automatic reports (appendix c); and at the very end, the index.

worked example

production line

a factory has one entrance guarded by a weighbridge over which all incoming and outgoing vehicles must pass. only one vehicle can move in this area at a time.

aluminium sheets are delivered to the factory in vans. the sheets are fed onto production lines, formed into cans, filled, capped and then placed in containers. the containers are removed by lorries.

the vans arrive at the factory periodically. once across the weighbridge (which takes two minutes in or out), each van goes to the rear of the factory to an unloading area where its load is removed with the assistance of a crane. the load of aluminium sheets fills three empty hoppers one by one. (full hoppers are then fitted onto the production lines.) each van then leaves, again passing over the weighbridge. to prevent congestion, at most four vans are allowed in the factory grounds at a time.

a pool of seven vans serves the factory. a res 'vanspaces' is used to limit the number in the factory grounds to 4 at any one time. unloading takes place when the crane and empty hoppers are available (an unloading, which fills three hoppers, may start even if only one or two are free; but the crane is only released by its owning van when three hoppers have been filled. the filling of each hopper takes about 5 minutes (normal, mean = 5, standard deviation = 1). after exiting, the van returns with a new load in about 108 minutes (98 + negexp(0.1)).

n.b. all the simulation timings are given in minutes.

resource diagram for class van

```
                    ------------<-----------------------
                    !                                  !
                    ! ----------<----------------      !
                    ! !                         !      !
                    ! ! --------<---------      !      !
                    ! ! !                !      !      !
                ----------               !      !      !
                ! enter  !               !      !      !
                ----------               !      !      !
                    !   !                !      !      !
                    !   -------->------- !      !      !
                    !                  ! !      !      !
                    !   ----<----      ! !      !      !
                    !   !       !      ! !      !      !
                ----------      !      ! !      !      !
                !  get   !      !      ! !      !      !
                ! crane  !      !      ! !      !      !
                ----------      !      ! !      !      !
                    !           !      ! !      !      !
                    !           !      ! !      !      !
         ---->----  !   --<--   !      ! !      !      !
         !       !  !   !   !   !      ! !      !      !
 empty !   !    ----------  !  +++     +++     +++     !
hoppers! 5 !    ! unload !  ! + 1 +   + 1 +   + 4 +    !
       -----    ----------  !  +++     +++     +++     !
                 !  !   !   ! crane   weigh-   van     !
         ----<----  !   -----   !     bridge  spaces   !
         !          !           !      ! !      !      !
  full !   !    ----------      !      ! !      !      !
hoppers! 3 !    ! return !      !      ! !      !      !
       -----    ! crane  !      !      ! !      !      !
                ----------      !      ! !      !      !
                    !   !       !      ! !      !      !
                    !   ---->----      ! !      !      !
                    !                  ! !      !      !
                    !   --------<------- !      !      !
                    !   !                !      !      !
                ----------               !      !      !
                ! leave  !               !      !      !
                ----------               !      !      !
                    ! ! !                !      !      !
                    ! ! --------->---------      !      !
                    ! !                         !      !
                    ! ---------->----------------      !
                    !                                  !
                    ------------>-----------------------
```

```
entity class van;
begin
  integer k;
enter:
  vanspaces.acquire(1);
  weighbridge.acquire(1);
  hold(2.0);
  weighbridge.release(1);
unload:
  crane.acquire(1);
  for k := 1 step 1 until 3 do
  begin
    emptyhoppers.take(1);
    hold(fill.sample);
    fullhoppers.give(1);
  end;
  crane.release(1);
leave:
  weighbridge.acquire(1);
  hold(2.0);
  weighbridge.release(1);
  vanspaces.release(1);
nextload:
  hold(98.0 + nexttrip.sample);
  repeat;
end***van***;
```

a full hopper fits onto a production line (of which there are five). the aluminium sheets are removed from the hopper and processed one by one. as the sheets pass down the line, they are formed into cans, filled with liquid x and capped. it takes two hoppers to fill one container. if all goes smoothly, the processing time per hopper is 25 minutes.

the containers are loaded onto articulated trucks. the trucks wait outside the factory until a loading bay is free. they take three minutes to cross the weighbridge (in and out) and then manoeuvre into a loading bay. when the lorry is loaded, it departs via the weighbridge.

lorries arrive roughly every 10 minutes (negexp(1/10)). they enter the factory grounds when they have a bay (there are 6 bays in the model) and the weighbridge. once in, they accept two containers and then leave.

resource diagram for class lorry

```
  ----->------     ------<-----
  !          !     !          !
  !         ---------         !
  !         ! enter !         !
  !         ---------         !
  !             !  !          !
  !             !  ------>--- !
  !             !           ! !
  !            ***          ! !
  !            *** bayq     ! !
  !            ***          ! !
  !                         ! !
 +++                        +++ weigh-
+ 6 + bays                 + 1 +
 +++                        +++ bridge
  !                         ! !
  !                         ! !
  !                         ! !
  !                         ! !
  !                ------<--- !
  !                !          !
  !         ---------         !
  !         ! leave !         !
  !         ---------         !
  !          !     !          !
  -----<------     ------>-----
```

```
entity class lorry;
begin
  new lorry("lorry").schedule(nextlorry.sample);
enter:
  bays.acquire(1);
  weighbridge.acquire(1);
  hold(3.0);
  weighbridge.release(1);
load:
  bayq.wait;

exit:
  weighbridge.acquire(1);
  hold(3.0);
  weighbridge.release(1);
  bays.release(1);
end***lorry***;
```

when a hopper is put on the line, the plant starts producing cans.

the first can is ready ten minutes later. if there is no waiting container, production is halted but can continue without penalty when one arrives. after a further fifteen minutes, the hopper has to be replaced with another possible production line halt. twenty five minutes later the second hopper will have been emptied, but the last can will not arrive at the end of the production line until another five minutes have elapsed.

a second container load can be started on the production line immediately after the first if required, there being no need to wait for the final can of the first load to be ready before the first can of the second can be started.

resource diagram for class production

```
                             --------------<-------------
                             !                          !
                             !     ------<----          !
                             !     !         !          !
                      ---------------        !          !
                      ! start first !        !          !
                      !    batch    !        !          !
             bayq     ---------------        !          !
             *** coopt a     !               !          !
             ***==========   !               !          !
             ***  lorry  "   !               !          !
                         "   !               !          !
                         "   !         --<-- !          !
                      ---------------  !   ! !          !
                      !  load  and  !  !  !   Full     !
                      !  continue   !  !  ! 3 Hoppers  !
                      ---------------  !  -----         !
                       !     !         !                !
             ----<------     !         !                !
             !               !         !                !
             ! --<--         !     --<--                !
             ! !   !         !     !                    !
            !   !  !  ---------------                   !
            ! 5 !  !  !    second   !                   !
            -----  !  !    batch    !                   !
            empty  !  ---------------                   !
           hoppers !   ! "   !                          !
                   --<-- "   !                          !
                         "   !                          !
                schedule "   !                          !
              <===========   -------------->-------------
                lorry l.
```

```
entity class production;
begin
  ref(lorry)l;
firsthopper:
  fullhoppers.take(1);
  hold(10.0);
findtruck:
  l :- bayq.coopt;
  hold(15.0);
  emptyhoppers.give(1);
secondhopper:
  fullhoppers.take(1);
  hold(25.0);
  emptyhoppers.give(1);
  l.schedule(10.0);
  repeat;
end***production line***;
```

the driving program reads:

```
begin
  external class demos;
  demos
    begin
      ref(rdist)nextlorry, fill, nexttrip;
      ref(res)weighbridge, crane, bays, vanspaces;
      ref(bin)fullhoppers, emptyhoppers;
      ref(waitq)bayq;

      entity class van.............;
      entity class lorry...........;
      entity class production......;

      integer k;

      nextlorry    :- new negexp("next lorry", 0.05);
      fill         :- new normal("fill hopper", 5.0, 1.0);
      nexttrip     :- new negexp("van return", 0.1);
      weighbridge  :- new res("weighbridge", 1);
      crane        :- new res("crane", 1);
      bays         :- new res("bays", 6);
      vanspaces    :- new res("van spaces", 4);
      bayq         :- new waitq("await container");
      fullhoppers  :- new bin("full hoppers",  3);
      emptyhoppers :- new bin("empty hoppers", 5);
      new lorry("l").schedule(0.0);
      for k := 1 step 1 until 7 do
        new van("v").schedule((k-1)*14);
      for k := 1 step 1 until 5 do
        new production("p-line").schedule(0.0);
```

```
      hold(480.0);
    end;
end;
```

```
                        the automatic demos report


                         clock time =    480.000
**********************************************************************
*                                                                    *
*                              r e p o r t                           *
*                                                                    *
**********************************************************************


                        d i s t r i b u t i o n s
                        *************************

title       /   (re)set/   obs/type     /        a/         b/     seed
next lorry        0.000     41 negexp        0.100              33427485
fill hopper       0.000     74 normal        5.000     1.000  22276755
van return        0.000     24 negexp        0.100              46847980


                            r e s o u r c e s
                            *****************

title       /   (re)set/   obs/ lim/ min/ now/  % usage/ av. wait/qmax
weighbridge       0.000    121    1    0    1    65.417     1.168    4
crane             0.000     24    1    0    0    81.393     8.943    2
bays              0.000     33    6    0    0    88.570    12.900    4
van spaces        0.000     24    4    0    3    40.525     0.000    1


                                 b i n s
                                 *******

title       /   (re)set/   obs/init/ max/ now/ av. free/ av. wait/qmax
full hoppers      0.000     73    3    3    0     0.516     5.477    5
empty hopper      0.000     71    5    8    2     2.571     0.219    1


                          w a i t   q u e u e s
                          *********************

title       /   (re)set/   obs/ qmax/ qnow/ q average/zeros/  av. wait
await contai      0.000     38     4     0      0.315    27      3.977
await contai*     0.000     38     3     1      0.578    12      6.743


program execution time 2.94 seconds (5 garbage collections took  0.460
seconds).  run on 1979/1/20.
```

chapter 2

classes tab and reportq

to enable automatic reporting, user-generated data collection devices, random number generators, queues, and resources are individually placed into special reportqs on creation. each such type of demos facility is prefixed by tab which gives its objects the capability of joining a reportq.

```
        -----------
        ! reportq !
        !---------!
        ! last  --!---------------------->---------------
        ! first --!---->----                            !
        ! report  !        !                            !
        ! reset   !        !                            !
        -----------    ----------    ----------    ----------
                       !  tab   !    !  tab   !    !  tab   !
                       !--------!    !--------!    !--------!
                       ! next  -!--->! next  -!--->! next   !
                       ! report !    ! report !    ! report !
                       ! reset  !    ! reset  !    ! reset  !
                       ----------    ----------    ----------
```

reportq with three tabs

each type of demos facility has its own special reportq:

```
accumq   - for accumulate objects
binq     - for bin objects
condqq   - for condq objects
countq   - for count objects
distq    - for rdist (except empirical), idist, and bdist objects
empq     - for empirical objects
queueq   - for queue objects
regressq - for regression objects
resq     - for res objects
tallyq   - for tally objects
waitqq   - for waitq objects
```

and on creation, each tab object enters the appropriate reportq at the end (as its new last).

given ref(tab)t and ref(reportq)r, t.report prints the current status of t on the next line of outf, r.report prints the current status of each of its tab members on successive lines (from first through to last, i.e. the order in which they were created). the global routine report goes through the demos defined reportq's one by one and reports them if they are not empty. the algorithm is:

```
]ref(reportq)q;[

for q :- distq, empq, accumq, ... , condqq do
  if q.first =/= none then
  begin
    2 new lines;
    q.report;
  end;
```

(the mechanism for the global routine reset is very similar). report is called automatically by the system when closing down a simulation unless switched off by a call on noreport.

2.1 class tab

class tab is used as prefix to data collection devices (count, tally, accumulate, regression, and histogram), random stream generators (constant, empirical, erlang, negexp, normal, uniform, poisson, randint, and draw), resources (res and bin), and queues (queue, condq, and waitq). its main function is to define the common portion of its sub-classes: namely a distinguishing title, the capability of being chained in a reportq, a primitive reset routine, a primitive report routine, and writetrn - the common portion of their report routines.

```
                              tab
                               !
                               !
            -----------------------------------------------
            !                !           !                !
            !                !           !                !
     data collection       queues     resources      random number
         devices                                       generators

        chapter 3        chapter 7   chapter 8        chapter 4
```

```
              tab object : new tab("stream");

                     -------------------
                     !       tab       !
                     -------------------
                     ! title  "stream" !
                     ! obs           0 !
                     ! resetat     0.0 !
                     ! next       none !
                     ! join(r)         !
                     ! report          !
                     ! reset           !
                     ! writetrn        !
                     -------------------
```

outline

```
class tab(title); value title; text title;
         virtual: procedure report, reset;
begin
   integer obs;
   real resetat;
   ref(tab)next;
```

```
procedure join(r); ref(reportq)r;
procedure report;
procedure reset;
procedure writetrn;
```

```
actions:
   if title.length  12 then title :- title.sub(1,12);
   reset;
end***tab***;
```

actions

the actions of the class body curtail the length of title to 12 characters should it be longer, and call reset (remember that this is virtual). reset sets resetat to the current clock time (= object creation time in the first instance), and sets obs to zero.

attributes

text title is a user-supplied descriptive text, cut off at 12 characters if initially longer.

integer obs records the number of observations (since resetat).

real resetat initially records the time of creation of the object. it is updated to the current clock time by each call on reset. it reflects the start of the time interval over which observations have been collected.

ref(tab)next points to the next tab in this tab's reportq.

procedure join(r) places the tab object into the named reportq as last object in that reportq.

procedure report sends one line to outf. it simply calls writetrn and then outf.outimage. it is redefined in each demos sub-class of tab.

procedure reset sets obs to zero and resetat to time. this reset is usually redefined in a sub-class and reinitialises the object so that data collection etc. can start again over a fresh time period. this is useful for erasing results gathered during a 'cold start', or when results are to be tabulated over successive time periods. it is specified as virtual to enable easy redefinition and to simplify the writing of the routine reset local to class reportq.

procedure writetrn writes (to the standard outfile outf) a part line containing the tab's title (columns 1-12), reset time (resetat, columns 14-23), and the number of observations recorded since resetat (obs, columns 24-30). resetat is usually printed fixed point, but should its value be very small (less than 0.1) or too large (=

1000000.0) to fit into columns 14-23 in the chosen format, it is printed floating point.

2.2 class reportq

reportq objects are used to chain together like tab objects created by the user. the following reportqs are generated by demos:

```
accumq   - for accumulate objects
binq     - for bin objects
condqq   - for condq objects
countq   - for count objects
distq    - for rdist (except empirical), idist, and bdist objects
empq     - for empirical objects
queueq   - for queue objects
regressq - for regression objects
resq     - for res objects
tallyq   - for tally objects
waitqq   - for waitq objects
```

each time a fresh (sub-class of) tab object is created, it is entered into the appropriate reportq at the end. thus these objects will be reported in the order of their creation.

reportq essentially contains a linked list for tab objects and two routines reset and report, each of which scans the list tab object by tab object and calls the local reset or report routine belonging to the currently referenced tab object.

reportq object : new reportq("tab", x, y);

```
-------------------
!     reportq     !
!-----------------!
! h         "tab" !
! l1         == x !
! l2         == y !
! first           !
! last            !
! report          !
! reset           !
-------------------
```

outline

```
class reportq(h, l1, l2); value h; text h, l1, l2;
begin
  ref(tab)first, last;

  procedure report;
  procedure reset;

end***reportq***;
```

actions

none.

attributes

text h, text l1, and text l2 provide the headings for the actual reports (see also report below).

ref(tab)first references none (the reportq is empty) or the first tab in the reportq.

ref(tab)last references none (the reportq is empty) or the last tab in the reportq.

procedure report prints out a centred heading (h) underlined by asterisks, followed by a line consisting of l1 concatenated with l2. thereunder follow the individual tab reports usually line by line (the exceptions are histoq, empq, and waitqq).

procedure reset moves along the tabs, one by one one, and calls the reset procedure of each.

chapter 3

data collection devices

demos defines four data collection devices for recording profiles of input sequences (each of which is prefixed by tab):

class count for incidences, e.g. the number of ingots produced by a furnace.

class tally for time independent sequences, e.g. the average number of items bought by customers in a supermarket.

class histogram caters for the same type of input stream as tally. over and above the profile recorded by a tally object, a histogram object produces a graphic representation of the input stream.

class accumulate for time dependent input sequences, e.g. the average number of customers in a supermarket.

class regression for linear regressions, that is best fit (in the least squares sense) linear relation between say the time spent queueing and the service time.

```
                         tab
                          !
                          !
       ----------------------------------------
       !         !         !         !        !
       !         !         !         !        !
  accumulate   count     tally   histogram regression
```

3.1 class count

count objects are used to record incidences. given the integer input sequence

n1, n2, ..., nm

a count objects records their sum n1 + n2 + ... + nm.

```
count object : new count("ingots")

        -------------------
        !      count      !
        -------------------
        ! title  "ingots" !
        ! virtual:report  !
        ! obs           0 !
        ! resetat     0.0 !
        ! next       none !
        ! join(r)         !
        ! reset           !
        ! writetrn        !   tab level
        -------------------
        ! report          !
        ! update(v)       !   count level
        -------------------
```

outline

```
tab class count;
begin
  procedure report;
  procedure update(v); integer v;

actions:
  join(countq);
end***count***;
```

actions

on generation, a count object executes the actions of its tab prefix (which curtail title to 12 characters, if longer, and then call reset). then it enters the reportq countq at the end.

attributes

(for title, obs, resetat, next, join(r), reset, and writetrn, see tab, page 2-4).

procedure report sends one line to outf:

title in columns 21-32

resetat in columns 34-43

obs in columns 44-50

procedure update(v) increments obs by v. note that this is a slightly different usage for obs than in other tab subclasses.

example of use

```
        ref(count)c;
c :- new count("ingots");
```

c is to be used to count the number of ingots created by furnaces in a steel mill simulation.

```
entity class furnace;
begin
  produce the next ingot;
  c.update(1);
  repeat;
end***furnace***;
```

typical report

```
           c o u n t s
           ***********

title       /   (re)set/   obs
ingots            0.000     47
```

3.2 class tally

a tally object is used to record a profile of a sequence of real values. given the real input sequence

x1, x2, ..., xm

a tally object records the number of observations (m), their sum so far (x1 + x2 + ... + xm), the sum of their squares so far (x1*x1 + x2*x2 + ... + xm*xm), and their range, that is the least and largest input values so far. from this data, it is simple to compute their mean and give an estimate for their standard deviation.

```
tally object : new tally("q times");

                   -------------------
                   !      tally      !
                   -------------------
                   ! title "q times" !
                   ! virtual:report  !
                   ! virtual:reset   !
                   ! obs           0 !
                   ! resetat     0.0 !
                   ! next       none !
                   ! join(r)         !
                   ! writetrn        !   tab level
                   -------------------
                   ! sum         0.0 !
                   ! sumsq       0.0 !
                   ! min         0.0 !
                   ! max         0.0 !
                   ! report          !
                   ! reset           !
                   ! update(v)       !   tally level
                   -------------------
```

outline

```
tab class tally;
begin
  real sum, sumsq, min, max;

  procedure report;
  procedure reset;
  procedure update(v); real v;

actions:
```

```
   join(tallyq);
end***tally***;
```

actions

on generation, a tally object executes the actions of its tab prefix (which curtail title to 12 characters, if longer, and then call reset). then it enters the reportq tallyq at the end.

attributes

(for title, obs, resetat, next, join(r), writetrn see tab, page 2-4).

real sum records the sum of the input values since time = resetat.

real sumsq records the sum of the squares of the input values since time = resetat.

real min records the least input value since time = resetat.

real max records the largest input value since time = resetat.

e.g. given the update sequence

1.0 4.0 9.0

obs = 3, sum = 14.0, sumsq = (1.0*1.0 + 4.0*4.0 + 9.0*9.0) = 98.0, min = 1.0, and max = 9.0.

procedure report sends one line to outf containing

title/resetat/obs/mean/est. stan. dev./min/max

```
where mean       = (x1+x2+...+xm)/obs      obs > 0
                 = undefined               obs = 0
and est.stan.dev = sqrt(variance)          obs > 1
                 = undefined               obs<= 1
    variance     = (obs*sumsq-sum*sum)
                   /(obs*(obs-1))          obs > 1
```

the object's title in columns 1-12.

the value of resetat in columns 14-23.

the number of observations recorded (obs) in columns 24-30.

the mean of the input values since resetat in columns 31-40.

an estimate for the standard deviation of the input values since resetat in columns 41-50.

the least input value min in columns 51-60.

the largest input value max in columns 61-70.

n.b. if obs = 0, columns 31-70 are skipped; if obs = 1, columns 41-50 are skipped.

procedure reset sets obs, sum, sumsq, min, max to zero, and resetat to time.

procedure update(v) increments obs by 1, adds v to sum, v*v to sumsq, sets min to the smaller of (v, min), and max to the larger of (v, max).

example of use

```
ref(tab)t;
t :- new tab("thru times");
```

t is to be used to record the average through times of customers in a supermarket.

```
entity class customer;
begin
  real arrivaltime;
  arrivaltime := time;
  do shopping;
  t.update(time-arrivaltime);
end***customer***;
```

typical report

```
                    t a l l i e s
                    *************

title       /   (re)set/   obs/  average/est.st.dv/  minimum/  maximum
thru times        0.000   1000    10.094     1.995     3.396    16.811
```

3.2.1 class notally

class notally has the declaration

```
tally class notally;;
```

the sole use in demos for such objects lies in class histogram where it is essential that the local 'tally' object is not a member of tallyq. since the class body actions of tally include

```
if not(this tally in notally) then join(tallyq);
```

tally objects enter tallyq but notally objects do not.

3.3 class histogram

a histogram object records a rough profile of a sequence of real values, xj, by asking in advance for their (expected) lower bound, lower, and upper bound, upper; and also for the number of recording cells, ncells. the range lower -> upper is then divided into ncells cells, each of the same width (upper-lower)/ncells. when an x value is recorded, the appropriate cell incidence count is incremented by 1. underflow values, xj < lower, and overflow values, xj >= upper, are recorded separately.

for example, after

```
h :- new histogram("q times", 2.0, 20.0, 9);
```

h will expect update values in the range 2.0 <= x < 20.0 and set up 9 recording divisions (plus one extra for underflow and one extra for overflow). input values are then grouped into cells with ranges

```
     <2.0, ]2.0->4.0), ]4.0->6.0),.., ]18.0->20.0), ]>=20.0
  underflow     1          2               9         overflow
```

the notation]a->b) means a <= x < b.

```
histogram object : new histogram("q times", 2.0, 20.0, 9);

           -------------------
           !    histogram    !
           -------------------
           ! title "q times" !
           ! virtual:report  !
           ! virtual:reset   !
           ! obs           0 !
           ! resetat     0.0 !
           ! next       none !
           ! join(r)         !
           ! writetrn        !   tab level
           -------------------
           ! lower       2.0 !
           ! upper      20.0 !
           ! ncells        9 !
           ! width       2.0 !
           ! table       ---!------
           ! limit        10 !  !  !
           ! myt==new notally!  !  !
           ! report          !  !  !
           ! reset           !  !  !
           ! update(v)       !  !  ! histogram level
           -------------------  ----
```

outline

```
tab class histogram(lower, upper, ncells);
        real lower, upper; integer ncells;
begin
  real width;
  integer array table(0:ncells+1);
  integer limit;
  ref(notally)myt;

  procedure report;
  procedure reset;
  procedure update(v); real v;

actions:
  if upper <= lower or ncells < 1 then error;
  width := (upper-lower)/ncells;
  limit := ncells+1;
  myt   :- new notally(title);
  join(histoq);
end***histogram***;
```

actions

on generation, an histogram object executes the actions of its tab prefix (which curtail title to 12 characters, if longer, and then call reset). a simula run time error is caused if ncells < -1; a demos error is given if ncells = -1 or 0. a check is made on the expected range of values. if this is illegal (upper <= lower), then a warning is given and lower is set to 0.0, upper to 100.0. width is set to the cell width, and limit to ncells plus 1. both the latter are used in report. then a notally object is created referenced by myt. (as myt is a notally object, it is not a member of tallyq.) then the histogram object enters the reportq histoq at the end.

attributes

(for title, obs, resetat, next, join(r), writetrn see tab, page 2-4).

real lower is interpreted as the lower expected bound for the input stream.

real upper is interpreted as the expected upper bound for the input stream.

integer ncells is interpreted as the number of recording cells;

ncells >= 1.

real width is set to the cell width by the class body actions. it is used in report and update.

integer limit is set to ncells+1 by the class body actions; it is used often in report.

integer array table(0:ncells+1) is the table in which the incidences are recorded. values in range go in cells 1-ncells, underflow values in cell 0, and overflow values in cell limit.

ref(notally)myt references a local notally object (see page 3-6). this is updated each time the histogram object is updated and thus makes available such useful information as the average, estimated standard deviation, and the actual range of the input values. should overflow and/or underflow occur, these minimum and maximum values enable more appropriate histogram bounds to be set next time. myt is reported and reset when the histogram is reported or reset respectively.

procedure report sends a report to outf. it first calls myt.report and then gives a 'picture' of the recorded values using the information stored in table. basically there are ncells+2 lines of output: one for underflow, one for each of the ncells 'useful' cells, and one line for overflow.

```
procedure report;
begin
  integer i;
  print heading;
  call myt.report;
  if obs = 0 then
  begin
    print no observations recorded;
  end else
  begin
    for i := 0 step 1 until limit do
    begin
      print line for cell i;
      if no more to print then goto finish;
    end;
finish:
  end;
end***report***;
```

on each line, we give the

cell number in columns 1-4.

the number of incidences (table(i)) in columns 15-20.

the frequency (as a %, 100.0*table(i)/obs) in columns 21-28.

the cumulative frequency (as a %) in columns 29-36, and then, in columns 41-70, a line of asterisks representing the relative frequency. the maximum number of incidences is accorded 30 asterisks; the remainder are reduced proportionately (rounded). zero entries are blanked off. entries which are non-zero but too small to warrant a single asterisk are represented by a dot '.' in column 41 (see example below). should the initial estimate of the histogram range be poor, and more than three rows at the end of the table be empty, then these last empty rows are skipped (see example below),

procedure reset sets obs to zero, sets each table entry to zero, resetat to time, and then calls myt.reset.

procedure update(v) increments obs by 1, locates the appropriate cell cell, and then increments table(cell) by 1. it also calls myt.update.

example of use

```
ref(histogram)h;
h :- new histogram("thru times", 5.0, 105.0, 20);
```

h is to be used to record the through times of customers in a supermarket.

```
entity class customer;
begin
  real arrivaltime;
  arrivaltime := time;
  do shopping;
  h.update(time-arrivaltime);
end***customer***;
```

typical report

given overleaf. it displays the following features:

underflow as recorded in cell 0.

cell 7 does not warrant an asterisk on the proportion 301 observations => 30 asterisks, but is non-zero. accordingly, it has been rewarded with a dot.

no observations have been recorded for cells 8-20 at the end of the table. these are skipped in the report ("***rest of table empty***"). holes (of size at least 3 lines) at the end of the table are always skipped; in the middle or at the beginning, they are always printed.

the summary records figures for the minimum (= 0.126) and the maximum (= 37.981) values of the update sequence, and suggests that a better generating statement for h would thus be

```
h :- new histogram("thru times", 0.0, 40.0, 10);
```

```
                         h i s t o g r a m s
                         *******************

                            s u m m a r y

title         /   (re)set/   obs/  average/est.st.dv/  minimum/  maximum
thru times          0.000   1000    18.404     7.016     0.126    37.981

cell/lower lim/    n/   freq/  cum %
                                       i-------------------------------
   0 -infinity    35    0.04    3.50   i***
   1     5.000    78    0.08   11.30   i********
   2    10.000   184    0.18   29.70   i******************
   3    15.000   260    0.26   55.70   i**************************
   4    20.000   301    0.30   85.80   i******************************
   5    25.000    97    0.10   95.50   i**********
   6    30.000    42    0.04   99.70   i****
   7    35.000     3    0.00  100.00   i.

                                             ***rest of table empty***

                                       i-------------------------------
```

3.4 class accumulate

an accumulate object is used to record a profile of a time dependent variable, e.g. average queue length. to do so, it records the time integral of a sequence of real values, xj interpreted by a step function in simulated time. that is, given the real input sequence x1, x2, ... , xm where xj was recorded at time tj, an accumulate object builds up the time integral as depicted below:

```
!
!         ----
!         !  !   ---
!  -------!  !   !//
!  !      !  !---!//
!  !      !  !   !//
!  X1    X2X3 X4
--------------------
   t1     t2 t3  t4
```

the area under the curve is collected in real sumt. it is sigma xj * (tj+1 - tj), i.e. the value xj is assumed to hold between clock times tj -> tj+1. the initial values when reset is called are t0 = time (resetat), and x0 = 0.0. the average x value is computed from sumt by dividing through by the time span (time - resetat). not quite - we must also take into account an end correction (the shaded area in the figure above). in much the same way, an estimate of the standard deviation of the xs is build up using sumsqt. also recorded are the number of observations (m), and the minimum and maximum of the xj.

```
            accumulate object : new accumulate("qlength");

                        -------------------
                        !    accumulate   !
                        -------------------
                        ! title "qlength" !
                        ! virtual:report  !
                        ! virtual:reset   !
                        ! obs           0 !
                        ! resetat     0.0 !
                        ! next       none !
                        ! join(r)         !
                        ! writetrn        !   tab level
                        -------------------
                        ! sumt        0.0 !
                        ! sumsqt      0.0 !
                        ! min         0.0 !
                        ! max         0.0 !
                        ! lasttime    0.0 !
                        ! lastv       0.0 !
```

```
! report          !
! reset           !
! update(v)       !   accumulate level
------------------
```

outline

```
tab class accumulate;
begin
  real sumt, sumsqt, min, max, lasttime, lastv;

  procedure report;
  procedure reset;
  procedure update(v); real v;

actions:
  join(accumq);
end***accumulate***;
```

actions

on generation, an accumulate object executes the actions of its tab prefix (which curtail title to 12 characters, if longer, and then call reset). then it enters the reportq accumq at the end.

attributes

(for title, obs, resetat, next, join(r), writetrn, see tab, page 2-4).

real sumt holds the time integral of the input values since resetat. this it does in 'one-behind-fashion' by adding in lastv*(time-lasttime) when update is called. see the figure on the previous page and its explanation as to why.

real sumsqt holds the time integral of the squares of the input values since resetat. this it does in 'one-behind-fashion' by adding in lastv**2*(time-lasttime) when update is called.

real min holds the least sample value since resetat, or zero if no observations have been recorded.

real max holds the largest sample value since resetat, or zero if no observations have been recorded.

real lasttime holds the time of the last update, or resetat if no observations have been recorded.

real lastv holds the last input value recorded. initially zero, it is not altered by subsequent calls on reset.

procedure report prints on one line to outf a profile of data gathered in the time interval resetat - time.

the object's title in columns 1-12.

the value of resetat in columns 14-23.

the number of observations recorded (obs) in columns 24-30.

the average input value sumt/(time-resetat) - with an end correction - in columns 31-40.

an estimate of the time weighted standard deviation of the input values sqrt(sumsqt/(time-resetat) - average**2) - with an end correction - in columns 41-50.

the least input value min in columns 51-60.

the largest input value max in columns 61-70.

n.b. if obs = 0 (no values have been recorded), columns 31-70 are skipped.

procedure reset sets obs, sumt, sumsqt, min, max are set to zero. resetat and lasttime are set to time. note that lastv remains unaltered.

procedure update(v) increments obs by 1, sumt by lastv*(time-resetat), sumsqt by lastv**2*(time-lasttime), and updates min and max if necessary (careful if it is the first observation). lastv is set to v, and lasttime to time.

example of use

```
     ref(accumulate)q;
q :- new accumulate("no. in shop");
```

q is to be used to maintain data on the average number of customers in a supermarket. the current number in the supermarket is maintained in a global integer n.

```
]integer n;[

entity class customer;
begin
  n := n+1;
  q.update(n);
  do shopping;
  n := n-1;
  q.update(n);
end***customer***;
```

warning: note the order; we set n first and then update q. remember that q.update(v) does not work with v and time but increments sumt and sumsqt using lastv and lasttime. in some other simulation packages this is not so, and their equivalents of accumulate would reverse this order.

typical report

```
                        a c c u m u l a t e s
                        *********************

title       /   (re)set/   obs/  average/est.st.dv/  minimum/  maximum
no. in shop       0.000   1000     5.112     1.993     0.000    12.000
```

3.5 class regression

a regression object is used to record data on a sequence of input pairs of real values (x1, y1), (x2, y2), ..., (xn, yn). x is treated as the independent variable, and when report is called it finds the best straight line approximation in the least squares sense

y = a0 + a1*x

the line passes through (xbar, ybar) and minimises

d1**2 + d2**2 + ... + dn**2

(for example, see the figure below for n = 6)

```
  !            D4/
  !            ! /
  !            !/
  !            /
  !           /
  !     d2!  /!
  !       ! / D5
  !       !/
  !    d6!/
  !      /
  !     /
  ! d1!/!
  !   / !
  !  /  !
  ! /   !
a0!/    D3
  !
  !
  !----------------
```

intercept = a0
slope = a1
passes through (xbar, ybar)

the resulting line is called the regression of y on x; (y is estimated from x).

```
regression object : new regression("response", "qlength");

                   -------------------
                   !    regression   !
                   -------------------
                   ! title"response" !
                   ! virtual:report  !
                   ! virtual:reset   !
                   ! obs           0 !
                   ! resetat     0.0 !
                   ! next       none !
                   ! join(r)         !
                   ! writetrn        !   tab level
                   -------------------
                   ! title2"qlength" !
                   ! x           0.0 !
                   ! y           0.0 !
                   ! xx          0.0 !
                   ! xy          0.0 !
                   ! yy          0.0 !
                   ! report          !
                   ! reset           !
                   ! update(vx, vy)  !   regression level
                   -------------------
```

outline

```
tab class regression(title2); value title2; text title2;
begin
  real x, y, xx, xy, yy;

  procedure report;
  procedure reset;
  procedure update(vx, vy); real vx, vy;

actions:
  if title2.length > 12 then title2 :- title2.sub(1, 12);
  join(regressq);
end***regression***;
```

actions

on generation, an regression object executes the actions of its tab prefix (which curtail title to 12 characters, if longer, and then call reset). its own actions curtail the length of title2 to 12 characters, if longer, and then it enters the reportq regressq at the end.

attributes

(for title, obs, resetat, next, join(r), writetrn, see tab, page 2-4).

text title (and with it, text title2) is used in a different sense from usual with tabs. they are used in the reports to explain what is being correlated with what. in a regression of y on x, title explains the x's, title2 the y's.

real x holds the sum of the x's since time = resetat.

real y holds the sum of the y's since time = resetat.

real xx holds the sum of the squares of the x's since time = resetat.

real xy holds the sum of the x*y products since time = resetat.

real yy holds the sum of the squares of the y's since time = resetat.

procedure update(vx, vy) increments obs and the 5 variables above in the obvious manner.

procedure reset sets obs and the 5 variables above to zero, and notes the time of the reset in resetat.

procedure report has a different format from the other data collection reports as there is too much information to print to be crammed onto one line. a typical report occupies 9 lines (several of them blank), e.g.

```
                    regression of 'r1y' upon 'r1x'

                 (re)set/    obs/     xbar/     ybar
                   0.000      50    22.950    25.500

 res.st.dev/ est.reg.coeff/ intercept/ st.dev.reg.coeff/ corr.coeff
      0.000          1.111     -0.000             0.000      1.000
```

the first line states what is regressing against what (using title for the x's and title2 for the y's), and is centered on the middle of the line. line 3 is a heading line, and line 4 prints the values of the last reset time, number of observations, and the average values for the x's and the y's. line 6 is another title line, and line 7 prints the values of the residual standard deviation, an estimate of the regression coefficient, the intercept on the y axis, the standard deviation of the regression coefficient assuming a student t-distribution with 2 degrees of freedom, and the correlation coefficient. the second, fifth, eighth and ninth lines are blank.

if obs <= 5, any analysis is deemed worthless and instead the text "*** insufficient data ***" is printed, e.g.

```
                regression of 'r5y' upon 'r5x'

             (re)set/    obs/     xbar/     ybar
               0.000      0

                 *** insufficient data ***
```

if the x values or the y values are the same (or very close together), then straight line data has been supplied; if both are constant, we have a point. a typical 'point' report is

```
                regression of 'r4y' upon 'r4x'

             (re)set/    obs/     xbar/     ybar
               0.000      50     1.000     1.000

                    ***degenerate data***
                  x = constant =      1.000
                  y = constant =      1.000
```

example of use

programs queue to use a cpu. we wish to compute the regression of the response times against the service times.

```
ref(regression)t;
ref(res)cpu;

t :- new regression("service time", response time");
cpu :- new res("cpu", 1);

entity class program;
begin
  real entry, st;
  new program("p").schedule(next.sample);
  entry := time;
  cpu.acquire(1);
  st := service.sample;
  hold(st);
  cpu.release(1);
  t.update(st, time-entry);
```

```
end***program***;
```

typical report

```
                 regression of 'response tim' upon 'service time'

                       (re)set/    obs/     xbar/     ybar
                         0.000      50     4.096     4.085

   res.st.dev/ est.reg.coeff/ intercept/ st.dev.reg.coeff/ corr.coeff
        0.737          0.053      3.870             0.184      0.042
```

chapter 4

random number generation

the basic generator used in demos is a lehmer generator reported and tested by downham and roberts]downham, 5[. it is

uk+1 = 8192*uk mod 67099547

by splitting 8192 into 32*32*8, the generator can be coded as

```
for k := 32, 32, 8 do
begin
  u := k*u;
  if u >= 67099547 then u := u - u//67099547*67099547;
end;
```

which will not cause overflow on a 32 bit machine since 67099547 < 67108864 = 2**26 and 32 = 2**5.

when it comes to generating well-spread-seeds automatically, we take advantage of a neat trick described by mats ohlin in]ohlin, 7[. note that if

uk+1 = a*uk mod m,

then

uk+2 = a*uk+1 mod m,
= a*a*uk mod m,
uk+3 = a*uk+2 mod m,
= a*a*a*uk mod m, etc.

it follows (see fuller]fuller, 6[), that if we use a**r as multiplier, we move through the basic cycle in steps of r at a time. not all values of r are suitable as the period may be drastically shortened through an unfortunate choice (again see fuller]fuller, 6[). we have chosen r = 120633 which has the multiplier 8192**120633 mod 67099547 = 36855 = 7*13*15*27. again this can be coded in simula so as not to give overflow on a 32 bit machine. the period of this (seed) generator is also 67099546.

4.1 well-spread-seeds

well-spread-seeds are automatically generated in demos by calls on zyqnextseed which works in conjunction with the global variable zyqseed.

outline

```
integer zyqseed;

integer procedure zyqnextseed;
begin
  integer k;
  for k := 7, 13, 15, 27 do
  begin
    zyqseed := zyqseed*k;
    if zyqseed >= zyqmodulo then zyqseed := zyqseed mod zyqmodulo;
  end;
  zyqnextseed := zyqseed;
end***zyqnextseed***;

procedure setseed(n); integer n;
begin
  if n < 0 then n := -n;
  if n >= zyqmodulo then n :=  n - n//zyqmodulo*zyqmodulo;
  if n = 0 then n := zyqmodulo//2;
  zyqseed := n;
end***setseed***;
```

attributes

integer zyqseed is initialised to 907 by the actions of the prefix demos and is the base for the 'well separated seed' generator. it may be reassigned by a call on the global procedure setseed. if you must alter zyqseed by direct assignment, make sure that 0 < zyqseed < zyqmodulo.

integer zyqmodulo is initialised by the actions of the prefix demos to 67099547. change it only if you change the basic random number generators.

integer procedure zyqnextseed works on zyqseed and from it produces the next well-spread-seed. zyqnextseed = 8192**120633 modulo 67099547 (=36855 = 7*13*15*27) * zyqseed modulo 67099547. zyqseed is updated to this new value which is the return value of the call.

procedure setseed(n); integer n; normally sets zyqseed to n, but if n lies outside the range]1, zyqmodulo-1[- which would cause zyqnextseed to overflow (or always return 0 if n zyqseed became 0) - n is manipulated into a safe value by

```
n := remainder(abs(n), zyqmodulo);
if n = 0 then n := zyqmodulo//2;
```

using setseed

initially zyqseed is set by demos to 907; thereafter every time a dist object is created (or a sub-class object), its u and ustart variables are set by calling zyqnextseed. each such call produces the next 'well-spread-seed' separated from the last by 120633 drawings on zyqsample.

each time a demos program is run, it generates well-spread-seeds in the predetermined order:

0	907
1	33427485
2	22276755
3	46847980
4	43859043
5	64042082
6	44366385
7	41357879
8	11320893
9	6528269
10	47478000

each of which has its 'own' portion of the basic cycle of length 120633 drawings. after 120633 drawings, the underlying rth distribution will start to overlap with the r+1st. this separation holds for over 275 distribution.

if you wish to vary these default values (except under 5 below), the following suggestions may be helpful.

1. alter the order of creation of your distributions.

2. make a call setseed to alter the base seed zyqseed in your program a) before generating any distributions, or b) in between generating distributions if you wish an abrupt change. b) is not recommended unless you have thought it through.

3. after creation of the object, make the call

```
d.setseed(n);
```

after creating d to change its start seed value. if antithetic drawings are wanted, use

```
d.antithetic := true;
```

since d.setseed(-d.u) will carefully replace d.u by itself. (for details of this setseed and antithetic, which are both local to class dist, see under dist, page 4-9.)

4. put code similar to the segment below in your program before generating any distributions:

```
]integer k;[

for k := 1 step 1 until 13 do
  zyqnextseed;
```

this causes the first 13 (arbitrarily chosen) default seeds to be skipped.

5. replace the body of integer procedure zyqnextseed by coding of your own.

4.2 distributions

this section describes the demos random number generators which are implemented in simula itself. as a result they are a little slower than their system defined equivalents. but they are portable, easier to use (in that the system provides the seeds and no parameters are needed on a call), and their profile and usage are automatically reported.

nine distributions are defined in demos:

returning real values we have:

constant(a) which always returns the same value a.

erlang(a, b) which returns a drawing from an erlang distribution with mean a and standard deviation a/sqrt(b). it can be thought of as the sum of b exponential distributions each with an expected value of a/b.

empirical(n) which reads in a cumulative probability distribution in the form of a table of 'size' n and returns samples from it.

negexp(a) which returns drawings from an exponential distribution with an arrival rate of a; that is, its expected value is 1/a.

normal(a, b) which returns drawings from a normal distribution with mean a and standard deviation b.

uniform(a, b) which returns drawings from a uniform distribution with lower bound a and upper bound b.

returning integer values we have:

poisson(a) which returns drawings from a poisson distribution with a mean of a.

randint(a, b) which returns drawings from a distribution randomly distributed amongst the integers a, a+1, ..., b.

returning boolean values we have:

draw(p) which returns true with probability p.

their implementation is according to the hierarchy:

```
                  tab
                   !
                   !
                  dist
                   !
                   !
         ---------------------
         !         !         !
         !         !         !
       rdist     idist     bdist
```

tab prefix enables each distribution to be placed in a reportq (empq for empiricals; distq for the rest). dist defines the common portion: an individual seed, u, their sampling mechanism, zyqsample, which works in conjunction with u, and a centralised report routine.

in addition:

rdist specifies a virtual real procedure sample, and is used to prefix constant, empirical, erlang, negexp, normal, and uniform.

idist specifies a virtual integer procedure sample, and is used to prefix poisson and randint.

bdist specifies as virtual a boolean procedure sample, and is used to prefix draw.

thus, in demos programs, where one usually only wishes to explicitly access the sample attribute explicitly, it is enough to declare all variables referencing sub-class of rdist objects with ref(rdist) variables; all sub-class of idist objects with ref(idist) variables; and all sub-class of bdist objects with ref(bdist) variables.

with each distribution we also give a typical histogram and, where it is useful, record the expected value, ex, of the particular random variable, and its variance vx = ex](x-m)[**2.

4.2.1 class dist

```
dist object : new dist("dist");

      -------------------
      !      dist       !
      -------------------
      ! title    "dist" !
      ! virtual:report  !
      ! obs           0 !
      ! resetat     0.0 !
      ! next       none !
      ! join(r)         !
      ! reset           !
      ! writetrn        !   tab level
      -------------------
      ! u      33427485 !
      ! ustart 33427485 !
      ! type          0 !
      ! antithetic false!
      ! setseed         !
      ! zyqsample       !
      ! zyqfail         !
      ! report          !   dist level
      -------------------
```

outline

```
tab class dist;
begin
  integer u, ustart, type;
  boolean antithetic;

  procedure setseed(n); integer n;
  real procedure zyqsample;
  procedure zyqfail;
  procedure report;

actions:
  u := ustart := zyqnextseed;
  if this dist in empirical then join(empq)
                            else join(distq);
end***dist***;
```

actions

besides the actions of its tab prefix (which curtail the length of title to 12 characters, if longer, and then call reset), a dist object copies the next system derived seed into u and ustart. ustart remains fixed (used in reports); u will be updated by each call on sample from its sub-class. then the object enters empq if it is in empirical; otherwise it joins distq. (a distinction is made because distq members have one line reports; empirical reports are considerably longer.)

each demos-defined distribution meant for direct use (this excludes dist, rdist, idist, and bdist) is given a type in the range 1 through 9.

attributes

(for title, obs, resetat, next, join(r), reset, and writetrn, see tab, page 2-4).

note that the inherited reset only affects obs and resetat. in particular it does not alter the value of u. if you wish to repeat the same sequence of drawings from a particular distribution d, do it explicitly by a coding sequence such as:

```
]ref(dist)d;[

d.report;
d.reset;
d.setseed(d.ustart);
```

integer u is used by the local sampling routine to 'remember' its whereabouts in the basic random number generator cycle. it is updated each time zyqsample is called. u is set by the class body actions to the next well-spread-seed value.

integer ustart retains the initial seed for this distribution. it is not altered by demos; but it is printed as part of the report routine. ustart is set by the class body actions to the next well-spread-seed value.

integer type. each demos-defined sub-class of dist is given a type-identifying index according to the table:

0	undefined
1	normal
2	uniform
3	erlang
4	randint
5	negexp
6	poisson
7	draw
8	constant
9	empirical

this index is used in the error reporting routine zyqfail, the routine report, the routine readdist, and also as an index into the array disttype. if you define your own sub-class to dist, remember to give it a distinguishing value for type and to extend the three routines zyqsample, zyqfail, and report below, and disttype appropriately.

boolean antithetic is normally false. if it is set to true, then calls on zyqsample return not u, but the value 1-u.

procedure setseed can be used to reset the automatic value assigned to u and ustart. it makes sure that the actual parameter n is valid, 1 <= n <= zyqmodulo.

real procedure zyqsample uses downham and roberts's]downham, 5[basic random number generator routine starting from u. on exit, a value, u, in the range (0.0, 1.0) is returned (note the final divide through by zyqmodulo). u points to the next entry point in the cycle, and obs has been incremented by 1. if antithetic is true, 1-u is returned.

procedure zyqfail is an error reporting routine which is called in sub-classes should the user attempt to create an object with unlawful parameters. the routine prints the cause of the error and the recovery made, e.g. in a randint object if the upper bound is less than the lower bound, the bounds are reversed.

procedure report serves for all distributions defined in demos except empirical (whose report occupies more than one line). it sends one line to outf consisting of:

title in columns 1-12.

the reset time resetat in columns 14-23.

the observations recorded since resetat, obs in columns 24-30.

the distribution type in columns 31-40.

the initial seed value, ustart, in columns 61-70.

and in columns 41-60, the distribution parameters whose meanings are:

distribution	a	b
constant	constant	
draw	probability	
erlang	mean	k = mean/(variance)**2
negexp	1/mean = arrival rate	
normal	mean	standard deviation
poisson	mean	
randint	lower bound	upper bound
uniform	lower bound	upper bound

the routine first calls writetrn which prints out the distributions title, reset time, and the observations recorded. it then prints disttype(type) to display its class name ("constant", etc.), and then switches on index type to enter code which prints out the distribution characteristics. n.b. note how at this outer level we can access attributes declared at inner levels by using

```
this dist qua class-name.attribute
```

the code branches come together again at exit where the seed start value, ustart, is printed, followed by a call on outf.outimage.

typical reports

```
                    d i s t r i b u t i o n s
                    *************************

title        /   (re)set/   obs/type     /        a/         b/     seed
load               0.000   1000 constant     50.000
waits              0.000   1000 normal       10.000      1.000  22276755
service            0.000   1000 uniform       1.000      3.000  46847980
bulb life          0.000   1000 erlang        0.750          3  43859043
next bus           0.000   1000 negexp        1.000             64042082
kicks              0.000   1000 poisson       0.600             41357879
throws             0.000   1000 randint           1          6  11320893
chance             0.000   1000 draw          0.400              6528269
```

for completeness, here is a typical empirical report as well:

```
                       e m p i r i c a l s
                       *******************

               title        /   (re)set/   obs/     seed
               weights             0.000   1000  33427485

                      k/  dist. x(k)/  prob. p(k)
                      1     58.00000     0.00000
                      2     63.00000     0.10000
                      3     68.00000     0.45000
                      4     70.00000     0.55000
                      5     75.00000     0.90000
                      6     80.00000     1.00000
```

n.b. histograms of these distributions are given in later sections of this chapter.

4.3 rdist and its sub-classes

the declaration of rdist is simply

```
dist class rdist; virtual : real procedure sample;;
```

it has an empty body, but makes the matching sample routine defined in its sub-classes available to a ref(rdist) variable. each sub-class of rdist contains parameters which specify the particular distribution, a real procedure sample which uses these parameters and the objects own procedure zyqsample. the initialising actions set type appropriately and check (where possible) for unlawful parameter values.

```
rdist object : new rdist("rdist");

-------------------
!      rdist      !
-------------------
! title   "rdist" !
! virtual:report  !
! obs           0 !
! resetat     0.0 !
! next       none !
! join(r)         !
! reset           !
! writetrn        !   tab level
-------------------
! u      22276755 !
! ustart 22276755 !
! type          0 !
! antithetic false!
! setseed         !
! zyqsample       !
! zyqfail         !
! report          !   dist level
-------------------
! virtual:sample  !   rdist level
-------------------
```

4.3.1 class constant

constant objects always return the same sample value. they are perhaps most useful in the early stages of model building where it sometimes helps to see more clearly what is going on if an actual distribution is replaced for a test run or two by its mean.

```
constant object : new constant("fixed", 35.0);

                  -------------------
                  !     constant    !
                  -------------------
                  ! title   "fixed" !
                  ! virtual:report  !
                  ! obs           0 !
                  ! resetat     0.0 !
                  ! next       none !
                  ! join(r)         !
                  ! reset           !
                  ! writetrn        !   tab level
                  -------------------
                  ! u      46847980 !
                  ! ustart 46847980 !
                  ! type          8 !
                  ! antithetic false!
                  ! setseed         !
                  ! zyqsample       !
                  ! zyqfail         !
                  ! report          !   dist level
                  -------------------
                  ! virtual:sample  !   rdist level
                  -------------------
                  ! a          35.0 !
                  ! sample          !   constant level
                  -------------------
```

outline

```
rdist class constant(a); real a;
begin
  real procedure sample;

actions:
  type := 8;
end***constant***;
```

actions

on generation, a constant object executes the actions of its prefixes: tab (which curtail title to 12 characters, if longer, and then call reset), dist (which set u and ustart, and enter the object into distq), and rdist (nil). the main class body actions set type to 8.

attributes

(for title, obs, resetat, next, join(r), reset, writetrn, see tab 2-4; for u, ustart, type, zyqsample, zyqfail, report, see dist, page 4-8:4-9).

real a is the constant value returned by each call on sample.

real procedure sample (which matches the specification at the rdist level) returns a and updates obs by one. n.b. it does not call zyqsample. for a constant distribution, ex = a, and vx = 0.0.

typical histogram

(1000 samples from a constant object with a = 50.0)

```
title       /   (re)set/   obs/  average/est.st.dv/  minimum/  maximum
load              0.000   1000    50.000     0.000    50.000    50.000

cell/lower lim/    n/   freq/  cum %
                                       i------------------------------
   0 -infinity     0    0.00    0.00   i
   1    45.000     0    0.00    0.00   i
   2    48.333  1000    1.00  100.00   i******************************
   3    51.667     0    0.00  100.00   i
   4    55.000     0    0.00  100.00   i
                                       i------------------------------
```

ex = 50.0; vx = 0.0; sigma = 0.0

4.3.2 class erlang

the erlang distribution is a generalisation of the exponential distribution. consider a sequence of k conversations, each of which has a mean life of a/k, taken one after another. what does this distribution look like?

the duration of the k conversations is the sum of k independent random variables u1, u2, ... , uk. their mean and their variance are:

```
ex(u) = ex(u1 + u2 + ... + uk) = k*(a/k) = a
vx(u) = vx(u1 + u2 + ... + uk) = k*(a/k)**2 = (a*a)/k
```

again, in situations where a machine has a vital part without which it cannot function, k-1 spares are usually provided as backup. suppose replacing a spent part takes a negligible time and that each part has a life time exponentially distributed with a mean life of a/k. the combined life of the part and the stockpile then has an erlang distribution with a mean life of a and a variance of (a*a)/k..]if on the other hand all k parts of a machine must function, the machine life is given by z = min(u1, u2, ... , uk), which is the exponential distribution with a mean life of a/k.[

an erlang object with parameters a and b implements an erlang distribution which sums b independent random variables, each with a mean life of a/b. its expected value is a and its variance (a*a)/b.

the erlang distribution is a subset of the very useful gamma distribution (erlang has integer b; gamma has real b). if a random variable is positive and unimodal, then the chances are that it is a member of the gamma family. by varying a and b (a > 0.0, b > 0), we can approximate the negative exponential function (b = 1), the normal distribution (a = 1/b and b large), and chi squared (a = 2*k, b = k for k degrees of freedom).

```
erlang object : new erlang("arrivals", 0.75, 3);


              -------------------
              !      erlang     !
              -------------------
              ! title"arrivals" !
              ! virtual:report  !
              ! obs           0 !
              ! resetat     0.0 !
              ! next       none !
              ! join(r)         !
              ! reset           !
              ! writetrn        !   tab level
              -------------------
              ! u      43859043 !
              ! ustart 43859043 !
              ! type          3 !
              ! antithetic false!
              ! setseed         !
              ! zyqsample       !
              ! zyqfail         !
              ! report          !   dist level
              -------------------
              ! virtual:sample  !   rdist level
              -------------------
              ! a          0.75 !
              ! b             3 !
              ! zyqab      0.25 !
              ! sample          !   erlang level
              -------------------
```

outline

```
rdist class erlang(a, b); real a; integer b;
begin
  real zyqab;
  real procedure sample;

actions:
  type  := 3;
  if a <= 0.0 then
  begin warning;
    a := if a < 0.0 then -a else 0.01;
  end;
  if b <= 0.0 then
  begin warning;
    b := if b < 0 then -b else 1;
  end;
```

```
  zyqab := a/b;
end***erlang***;
```

actions

on generation, a erlang object executes the actions of its prefixes: tab (which curtail title to 12 characters, if longer, and then call reset), dist (which set u and ustart, and enter the object into distq), and rdist (nil). then the main class body actions set type to 3; check that a is positive (replacing it by -a if negative and 0.01 if zero); that b is positive (replacing it by -b if negative and 1 if zero); and finally sets zyqab (which speeds up the calls on sample).

attributes

(for title, obs, resetat, next, join(r), reset, writetrn, see tab 2-4; for u, ustart, type, zyqsample, zyqfail, report, see dist, page 4-8:4-9).

real a represents the expected value of the erlang distribution.

real b represents the number of sub-components making up the erlang distribution = (mean/standard deviation)**2. each sub-component has the same expected value a/b.

real zyqab is set to a/b by the actions of the class body. it is used as an auxiliary by calls on sample. zyqab is the expected value of each of the b erlang sub-components.

real procedure sample does not use the formula given in the common base]4, page 104[. it uses the log of products rather than the sum of logs to return a value from an erlang distribution with mean a and standard deviation (a/sqrt(b)). care is taken to maintain obs correctly.

typical histogram

(1000 samples from an erlang object with a = 0.75, and b = 3)

```
title       /   (re)set/   obs/  average/est.st.dv/  minimum/  maximum
bulb life         0.000   1000     0.743     0.437 4.061e-02     3.003


cell/lower lim/    n/   freq/  cum %
                                       i------------------------------
   0 -infinity     0    0.00    0.00   i
   1     0.000   332    0.33   33.20   i***********************
   2     0.500   442    0.44   77.40   i******************************
   3     1.000   169    0.17   94.30   i***********
   4     1.500    38    0.04   98.10   i***
   5     2.000    16    0.02   99.70   i*
   6     2.500     2    0.00   99.90   i.
   7     3.000     1    0.00  100.00   i.
   8     3.500     0    0.00  100.00   i
   9     4.000     0    0.00  100.00   i
                                       i------------------------------
```

ex = 0.750; vx = 0.1875; sigma = 0.434

4.3.3 class empirical

empirical objects represent cumulative probability functions. they are used to represent data which has not been given a functional description. instead, the distribution is stored as size pairs of values in two real arrays: x for the function value, and p for the corresponding cumulative probability. for a cumulative probability distribution, the first element of p, p(1) = 0.0, and the last element of p, p(size), = 1.0. further, pk > pj for k > j; and x(k) >= x(j) for k > j.

to sample from such a distribution, we first call zyqsample to produce a random number 'p' uniformly distributed over (0, 1). the value p is interpreted as a probability and is used to locate that index k for which p(k-1) < p < p(k) (and such a k must be found since 0.0 < p < 1.0, p(k-1) < p(k), p(1) = 0.0, and p(size) = 1.0). the desired sample value is found from the x table by linear interpolation between x(k-1) and x(k).

for example, suppose we have a table of weights: there is a 10% chance of a weight being 58->63 kg, a 35% chance of a weight being 63->68 kg, a 5% chance of it being 68->69 kg, a 5% chance of it being 69->70 kg, a 35 chance of it being 70->75 kg, and a 10% chance of it being 75->80 kg. the distribution has the graph depicted below, data points are starred.

```
80!                                       *
  !                                       !
  !                                       !
  !                                       !
  !                                       !
75!                                   *   !
  !                                       !
  !                                       !
  !      sample result                    !
  !.........................x             !
70!                     *   .             !
  !                   *     .             !
  !                 *       .             !
  !                         .             !
  !                         .             !
65!                         .             !
  !                         .             !
  !   *                     .             !
  !                         .             !
  !                         .             !
60!                         .             !
  !                         .             !
58*---+---+---+---+---+---+---+---+---+---+
```

0.0 0.2 0.4 0.6 0.8 1.0

.

probability

0.65

it is tabulated as

index k	x(k)	p(k)
1	58.0	0.00
2	63.0	0.10
3	68.0	0.45
4	70.0	0.55
5	75.0	0.90
6	80.0	1.00

(why may the mid point reading be skipped?)

suppose zyqsample returns 0.65. then, using the data above, k = 5 and

```
sample := 70.0 + (75.0-70.0)*(0.65-0.55)/(0.90-0.55)
        =  71.428
```

the representation of stepped data, typified by 1, 2, 2, 4 with equal weights is not quite so easy. we cannot write

index k	x(k)	p(k)
1	1	0.000000
2	1	0.250000
3	2	0.250000
4	2	0.750000
5	4	0.750000
6	4	1.000000

as the linear interpolation formula requires p(k) > p(k-1) else we risk division by zero. we could tabulate the data as a sequence of flat steps separated by almost vertical rises. for example

index k	x(k)	p(k)
1	1	0.000000
2	1	0.250000
3	2	0.250001
4	2	0.750000
5	4	0.750001
6	4	1.000000

this is not quite the correct distribution, but could be near enough. most of the time, sample will return the expected (flat) values; occasionally it will return interpolated values from the steep inter-connections. we must be prepared to trap these, if necessary, in the context of the sampling call.

another way of doing the same thing is to have a base distribution, a randint 0->99 object or a uniform 0.0->100.0 object, and draw a percentage from it. then explicitly code the selection as an if-expression. for example,

```
]r100 :- new randint("0->99", 0, 99); integer r, s;[

r := r100.sample;
s := if r < 50 then 2 else if r < 75 then 1 else 4;
```

(s now holds the required sample value.)

example: extending a demos definition

finally we could extend demos by introducing a class specially designed for the purpose. the easiest way of doing this is to build upon empirical as below:

```
empirical class step;
begin
  real procedure sample;
  begin
    real q; integer k;
    q := zyqsample;
    k := 2;
    while p(k) <= q do
      k := k+1;
    sample := x(k);
  end***sample***;
end***step***;
```

appropriate data would be:

```
(type = "empirical", size = 4)

          1   0.00
          2   0.25
          4   0.75
          4   1.00
```

step objects, being in empirical, would automatically be entered into empq on creation and automatically be reported and reset with other emp objects in the same reportq.

```
empirical object : new empirical("emp", 5);

-------------------
!    empirical    !
-------------------
! title     "emp" !
! virtual:report  !
! obs           0 !
! resetat     0.0 !
! next       none !
! join(r)         !
! reset           !
! writetrn        !   tab level
-------------------
! u      64042082 !
! ustart 64042082 !
! type          9 !
! antithetic false!
! setseed         !
! zyqsample       !
! zyqfail         !   dist level
-------------------
! virtual:sample  !   rdist level
-------------------
! size          5 !
! x             --!----------------
! p             --!------     !//!
! sample          !  !//!     !//!
! report          !  !//!     !//!
! read            !  !//!     !//!   empirical
-------------------  !//!     !//!     level
                     !//!     ----
                     ----
```

outline

```
rdist class empirical(size); integer size;
begin
  real array x, p(1:size);

  real procedure sample;
  procedure report;
  procedure read;

actions:
  type := 9;
  if size = 1 then error;
  read;
end***empirical***;
```

actions

on generation, a empirical object executes the actions of its prefixes: tab (which curtail title to 12 characters, if longer, and then call reset), dist (which set u and ustart, and enter the object into empq), and rdist (nil). the main class body actions then set type to 9. a simula run time error results if size < 1; a demos error is caused if size = 1. if size > 1 then size pairs of values (x(j), p(j)) are read in from inf by calling up read.

attributes

(for title, obs, resetat, next, join(r), reset, writetrn, see tab 2-4; for u, ustart, type, zyqsample, zyqfail, report, see dist, page 4-8:4-9).

integer size is used to set the array bounds for the x and p arrays. if size < 1 then a simula run time error will result. a demos error results if size = 1, as then we have the inconsistent demands that p(1) = 0.0 and p(size) = 1.0. further the sample routine would also fail on array index out of range.

real array x(1:size) holds the function values, x(j). note that x(k) >= x(j) for k > j, 1 <= k,j <= size.

real array p(1:size) holds the probability coordinates. note that p(k) > p(j) for k > j, 1 <= k,j <= size.

real procedure sample returns a drawing from the distribution specified by the x and p arrays according to the algorithm:

```
              q := random probability;
         locate interval p(k-1) <= q <= p(k) containing q:
              k := 2;
              while p(k) <= q do
                k := k+1;
              sample := x(k-1) + fraction*(x(k)-x(k-1))

where         fraction := (q-p(k-1))/(p(k)-p(k-1))
```

procedure report sends several lines to outf. first the object's title, reset time, number of recorded observations since resetat and initial seed are recorded, and then the table values are printed. see typical report on page 4-10.

procedure read reads the table values from inf. it assumes that size > 1 has been given, and attempts to read size pairs of values (x(k), p(k)). it checks that p(1) = 0.0 and p(size) = 1.0; and replaces them appropriately if not. as the kth pair is read in (k = 2, 3, ..., size), read checks that

a) x(k) >= x(k-1). if not, a warning is given and the input value for x(k) is replaced by x(k-1).

b) 0 < p(k) < 1 and p(k) > p(k-1). if not, a warning is given and the input value for p(k) is replaced by min(p(k-1) + 0.001, 1.0).

thus read guarantees that p(1) = 0.0, p(k) > p(j), p(size) = 1.0, x(k) >= x(j) for k > j and 1 <= k,j <= size.

typical histogram

(1000 samples from the empirical object given on pages 4-10 and 4-18)

```
title       /   (re)set/   obs/  average/est.st.dv/  minimum/  maximum
weights           0.000   1000    68.853     4.993    58.038    79.960

cell/lower lim/    n/   freq/  cum %
                                      i------------------------------
   0 -infinity     0    0.00    0.00   i
   1    57.000    47    0.05    4.70   i*******
   2    60.000    66    0.07   11.30   i*********
   3    63.000   192    0.19   30.50   i***************************
   4    66.000   208    0.21   51.30   i*****************************
   5    69.000   182    0.18   69.50   i**************************
   6    72.000   213    0.21   90.80   i******************************
   7    75.000    59    0.06   96.70   i********
   8    78.000    33    0.03  100.00   i*****
   9    81.000     0    0.00  100.00   i
                                      i------------------------------
```

4.3.4 class normal

normal objects reflect a particular normal distribution with a user specified mean a and standard deviation b. b must be positive.

```
normal object : new normal("n", 10.0, 1.0);

                   -------------------
                   !     normal      !
                   -------------------
                   ! title       "n" !
                   ! virtual:report  !
                   ! obs           0 !
                   ! resetat     0.0 !
                   ! next       none !
                   ! join(r)         !
                   ! reset           !
                   ! writetrn        !   tab level
                   -------------------
                   ! u      44366385 !
                   ! ustart 44366385 !
                   ! type          1 !
                   ! antithetic false!
                   ! setseed         !
                   ! zyqsample       !
                   ! zyqfail         !
                   ! report          !   dist level
                   -------------------
                   ! virtual:sample  !   rdist level
                   -------------------
                   ! a          10.0 !
                   ! b           1.0 !
                   ! zyqu        0.0 !
                   ! zyqv        0.0 !
                   ! zyqeven   false !
                   ! sample          !   normal level
                   -------------------
```

normal is implemented according to an algorithm which requires two independent uniform numbers u and v, but from them yields a pair of independently distributed normal random variables x and y

```
x = sqrt(-2*ln(u))*cos(2*pi*v)
y = sqrt(-2*ln(u))*sin(2*pi*v)
```

with mean 0 and standard deviation 1. the sample values for a normal distribution with mean a and standard deviation b are a + b*x and a + b*y respectively.

outline

```
rdist class normal(a, b); real a, b;
begin
  real zyqu, zyqv;
  boolean zyqeven;

  real procedure sample;

actions:
  type := 1;
  if b < 0.0 then
  begin
    warning - b replaced by B!;
    b := -b;
  end;
end***normal***;
```

actions

on generation, a normal object executes the actions of its prefixes: tab (which curtail title to 12 characters, if longer, and then call reset), dist (which set u and ustart, and enter the object into distq), and rdist (nil). the main class body actions set type to 1, and if the standard deviation supplied is negative, issue a warning and replace b by -b.

attributes

(for title, obs, resetat, next, join(r), reset, writetrn, see tab 2-4; for u, ustart, type, zyqsample, zyqfail, report, see dist, page 4-8:4-9).

real zyqu, zyqv, zyqeven are auxiliary variables used by sample.

real a is the user specification of the mean.

real b is the user specification of the standard deviation.

real procedure sample returns a drawing from a normal distribution with mean a and standard deviation b. to take maximum advantage from the work done, on odd requests for sample, the algorithm takes two drawings of zyqsample (say, u and v) and stores sqrt(-2.0*ln(u)) in zyqu and 2.0*pi*v in zyqv. it then returns zyqu*sin(zyqv) as the value of sample. the next request (which is perforce even) returns zyqu*cos(zyqv) as the value of sample. care is taken to maintain obs correctly. for the normal object, ex = a, and

```
vx = b*b.
```

typical histogram

(1000 samples from a normal object with a = 10.0 and b = 1.0)

```
title       /   (re)set/   obs/  average/est.st.dv/  minimum/  maximum
weights           0.000   1000    10.045     1.009     7.092    13.899

cell/lower lim/    n/   freq/  cum %
                                     i------------------------------
   0 -infinity     0    0.00    0.00   i
   1     6.000     0    0.00    0.00   i
   2     7.000    17    0.02    1.70   i*
   3     8.000   141    0.14   15.80   i************
   4     9.000   319    0.32   47.70   i***************************
   5    10.000   356    0.36   83.30   i******************************
   6    11.000   138    0.14   97.10   i************
   7    12.000    25    0.03   99.60   i**
   8    13.000     4    0.00  100.00   i.
                                     i------------------------------
```

ex = 10.000; vx = 1.000; sigma = 1.000

4.3.5 class negexp

a negexp object implements an negative exponential distribution with a user specified arrival rate a.

```
negexp object : new negexp("neg", 0.1);

                   -------------------
                   !     negexp      !
                   -------------------
                   ! title     "neg" !
                   ! virtual:report  !
                   ! obs           0 !
                   ! resetat     0.0 !
                   ! next       none !
                   ! join(r)         !
                   ! reset           !
                   ! writetrn        !   tab level
                   -------------------
                   ! u      41357879 !
                   ! ustart 41357879 !
                   ! type          5 !
                   ! antithetic false!
                   ! setseed         !
                   ! zyqsample       !
                   ! zyqfail         !
                   ! report          !   dist level
                   -------------------
                   ! virtual:sample  !   rdist level
                   -------------------
                   ! a           0.1 !
                   ! sample          !   negexp level
                   -------------------
```

outline

```
rdist class negexp(a); real a;
begin
  real procedure sample;

actions:
  type := 5;
  if a <= 0.0 then
  begin
    warning;
    a := if a < 0.0 then -a else 0.001;
  end;
```

```
end***negexp***;
```

actions

on generation, a negexp object executes the actions of its prefixes: tab (which curtail title to 12 characters, if longer, and then call reset), dist (which set u and ustart, and enter the object into distq), and rdist (nil). the main class body actions set type to 5, and if a -1/mean- is non-positive, issue a warning and replace a negative a by -a and a zero a by 0.001.

attributes

(for title, obs, resetat, next, join(r), reset, writetrn, see tab 2-4; for u, ustart, type, zyqsample, zyqfail, report, see dist, page 4-8:4-9).

real a represents the mean arrival rate of the distribution, i.e. the expected value returned by sample is 1/a.

real procedure sample returns a drawing from a negative exponential distribution with mean arrival rate of a. it is quite simply

-ln(zyqsample)*a

the, expected value, ex, is 1/a; and the variance, vx, is 1/(a*a).

typical histogram

(1000 samples from a negexp object with a = 1.0)

```
title       /   (re)set/   obs/  average/est.st.dv/  minimum/  maximum
next bus          0.000   1000     1.043     1.032 1.829e-04     7.643

cell/lower lim/    n/   freq/  cum %
                                      i-----------------------------
   0 -infinity     0    0.00    0.00  i
   1     0.000   620    0.62   62.00  i*******************************
   2     1.000   237    0.24   85.70  i***********
   3     2.000    92    0.09   94.90  i****
   4     3.000    36    0.04   98.50  i**
   5     4.000     5    0.01   99.00  i.
   6     5.000     4    0.00   99.40  i.
   7     6.000     5    0.01   99.90  i.
   8     7.000     1    0.00  100.00  i.
   9     8.000     0    0.00  100.00  i
                                      i-----------------------------
```

ex = 1.000; vx = 1.000; sigma = 1.000

4.3.6 class uniform

a uniform object implements a uniform (rectangular) distribution with a user specified range, from the lower bound a up to the upper bound b (but not inclusive).

```
uniform object : new uniform("range", 5.0, 15.0);
```

```
-------------------
!     uniform     !
-------------------
! title   "range" !
! virtual:report  !
! obs           0 !
! resetat     0.0 !
! next       none !
! join(r)         !
! reset           !
! writetrn        !   tab level
-------------------
! u      11320893 !
! ustart 11320893 !
! type          2 !
! antithetic false!
! setseed         !
! zyqsample       !
! zyqfail         !
! report          !   dist level
-------------------
! virtual:sample  !   rdist level
-------------------
! a           5.0 !
! b          15.0 !
! zyqspan    10.0 !
! sample          !   uniform level
-------------------
```

outline

```
rdist class uniform(a, b); real a, b;
begin
  real zyqspan;
  real procedure sample;
actions:
  type := 2;
  if a > b then
  begin
    warning;
```

```
    a <-> b;
  end;
  zyqspan := b-a;
end***uniform***;
```

actions

on generation, a uniform object executes the actions of its prefixes: tab (which curtail title to 12 characters, if longer, and then call reset), dist (which set u and ustart, and enter the object into distq), and rdist (nil). the main class body actions set type to 2, and check that a < b. if not a warning is given and the bounds are swapped. then the local variable zyqspan is set to the interval width b-a (it is used to speed up calls on zyqsample).

attributes

(for title, obs, resetat, next, join(r), reset, writetrn, see tab 2-4; for u, ustart, type, zyqsample, zyqfail, report, see dist, page 4-8:4-9).

real a is the lower bound of the distribution, a < sample.

real b is the upper bound of the distribution, b > sample.

real zyqspan holds the width of the distribution interval, b-a, and is set by the class body actions.

real procedure sample returns a drawing from a distribution which is uniform over (a, b). it is 'a + zyqspan*zyqsample'. the expected value, ex, is (b+a)/2; and the variance, vx, is (b+a)**2/12.

typical histogram

(1000 samples from a uniform object with a = 1.0 and b = 3.0)

```
title       /   (re)set/   obs/  average/est.st.dv/  minimum/  maximum
service            0.000   1000     1.979     0.554     1.000     3.000

cell/lower lim/    n/   freq/  cum %
                                      i------------------------------
   0 -infinity     0    0.00    0.00   i
   1     1.000   112    0.11   11.20   i***********************
   2     1.250   125    0.13   23.70   i*************************
   3     1.500   148    0.15   38.50   i******************************
   4     1.750   121    0.12   50.60   i*************************
   5     2.000   144    0.14   65.00   i*****************************
   6     2.250   140    0.14   79.00   i****************************
   7     2.500   102    0.10   89.20   i*********************
   8     2.750   108    0.11  100.00   i**********************
```

```
9      3.000     0    0.00  100.00   i
                                     i------------------------------

          ex = 2.000; vx = 1/3 = 0.333; sigma = 0.577
```

4.4 idist and its sub-classes

the declaration of idist is simply

```
dist class idist; virtual : integer procedure sample;;
```

it has an empty body, but makes the matching sample routine defined in its sub-classes available to a ref(idist) variable. each sub-class of idist contains parameters which specify the particular distribution, a integer procedure sample which uses these parameters and the objects own procedure zyqsample. the initialising actions set type appropriately and check for unlawful parameter values where possible.

```
idist object : new idist("idist");

    -------------------
    !      idist      !
    -------------------
    ! title   "idist" !
    ! virtual:report  !
    ! obs           0 !
    ! resetat     0.0 !
    ! next       none !
    ! join(r)         !
    ! reset           !
    ! writetrn        !   tab level
    -------------------
    ! u       6528269 !
    ! ustart  6528269 !
    ! type          0 !
    ! antithetic false!
    ! setseed         !
    ! zyqsample       !
    ! zyqfail         !
    ! report          !   dist level
    -------------------
    ! virtual:sample  !   idist level
    -------------------
```

4.4.1 class poisson

a poisson object takes a single real parameter a, representing a mean random arrival time, and returns integer sample values which represent drawings from a poisson distribution with a mean of a.

```
poisson object : new poisson("kicks", 0.1);

                -------------------
                !     poisson     !
                -------------------
                ! title   "kicks" !
                ! virtual:report  !
                ! obs           0 !
                ! resetat     0.0 !
                ! next       none !
                ! join(r)         !
                ! reset           !
                ! writetrn        !   tab level
                -------------------
                ! u      47478000 !
                ! ustart 47478000 !
                ! type          6 !
                ! antithetic false!
                ! setseed         !
                ! zyqsample       !
                ! zyqfail         !
                ! report          !   dist level
                -------------------
                ! virtual:sample  !   idist level
                -------------------
                ! a           0.1 !
                ! sample          !   poisson level
                -------------------
```

outline

```
idist class poisson(a); real a;
begin
  real procedure sample;

actions:
  type := 6;
  if a <= 0.0 then
  begin
    warning;
    a := if a < 0.0 then -a else 0.001;
```

```
   end;
end***poisson***;
```

actions

on generation, a poisson object executes the actions of its prefixes: tab (which curtail title to 12 characters, if longer, and then call reset), dist (which set u and ustart, and enter the object into distq), and rdist (nil). the main class body actions set type to 6, and if a - 1/mean - is non-positive, issue a warning and replace a negative a by -a and a zero a by 0.001.

attributes

(for title, obs, resetat, next, join(r), reset, writetrn, see tab, page 2-4; for u, ustart, type, zyqsample, zyqfail, report, see dist, page 4-8:4-9).

real a represents the arrival rate; that is, the expected number of arrivals per unit time.

integer procedure sample returns drawings from a poisson distribution with mean a according to the formula specified in the common base]4, page 106[. note that since the routine takes m+1 calls on zyqsample (each of which updates obs by 1), obs is corrected in the routine just before the final end. the expected value, ex, is a; and the variance, vx, is also a.

typical histogram

(1000 samples from a poisson object with a = 0.6)

```
title       /   (re)set/   obs/  average/est.st.dv/  minimum/  maximum
kicks             0.000   1000     0.632     0.798     0.000     5.000

cell/lower lim/    n/   freq/  cum %
                                       i------------------------------
   0 -infinity     0    0.00    0.00   i
   1     0.000   530    0.53   53.00   i******************************
   2     1.000   340    0.34   87.00   i*******************
   3     2.000   105    0.11   97.50   i******
   4     3.000    19    0.02   99.40   i*
   5     4.000     5    0.01   99.90   i.
   6     5.000     1    0.00  100.00   i.
   7     6.000     0    0.00  100.00   i
                                       i------------------------------
```

ex = 0.600; vx = 0.600; sigma = 0.775

4.4.2 class randint

a randint object implements a random (integer) distribution with a user specified range, from the lower bound a up to the upper bound b (including both).

```
randint object : new randint("terminal", 1, 6);

                 -------------------
                 !     randint     !
                 -------------------
                 ! title"terminal" !
                 ! virtual:report  !
                 ! obs           0 !
                 ! resetat     0.0 !
                 ! next       none !
                 ! join(r)         !
                 ! reset           !
                 ! writetrn        !   tab level
                 -------------------
                 ! u      46802881 !
                 ! ustart 46802881 !
                 ! type          4 !
                 ! antithetic false!
                 ! setseed         !
                 ! zyqsample       !
                 ! zyqfail         !
                 ! report          !   dist level
                 -------------------
                 ! virtual:sample  !   idist level
                 -------------------
                 ! a             1 !
                 ! b             6 !
                 ! zyqab         6 !
                 ! sample          !   randint level
                 -------------------
```

outline

```
idist class randint(a, b); real a, b;
begin
  integer zyqspan;
  integer procedure sample;
actions:
  type  := 4;
  if a > b then
  begin warning;
```

```
    a <-> b;
  end;
  zyqspan := b-a+1;
end***randint***;
```

actions

on generation, a randint object executes the actions of its prefixes: tab (which curtail title to 12 characters, if longer, and then call reset), dist (which set u and ustart, and enter the object into distq), and rdist (nil). the main class body actions set type to 4 and then check that a < b. if not a warning is given and the bounds are swapped. then the local variable zyqspan is set to the interval width plus 1 (it is used to speed up the call on zyqsample).

attributes

(for title, obs, resetat, next, join(r), reset, writetrn, see tab 2-4; for u, ustart, type, zyqsample, zyqfail, report, see dist, page 4-8:4-9).

integer a is the lower bound of the distribution, a <= sample.

integer b is the upper bound of the distribution, b >= sample.

integer zyqspan holds the width of the distribution range plus 1 (zyqspan >= 1) and is set by the class body actions. it is used to speed up the calls on sample.

integer procedure sample returns a sample from a distribution which is random over the integers a, a+1, a+2, ..., b. it is 'a + entier(zyqspan*zyqsample)'. remember that 0.0 < zyqsample < 1.0. the expected value, ex, is (b+a)/2; the variance, vx, is ((b-a+1)**2-1)/12.

typical histogram

(1000 samples from a randint object with a = 1 and b = 6)

```
title       /   (re)set/   obs/  average/est.st.dv/  minimum/  maximum
throws            0.000   1000     3.505     1.691     1.000     6.000

cell/lower lim/    n/   freq/  cum %
                                      i------------------------------
   0 -infinity     0    0.00    0.00   i
   1     0.500   166    0.17   16.60   i****************************
   2     1.500   157    0.16   32.30   i**************************
   3     2.500   171    0.17   49.40   i*****************************
   4     3.500   178    0.18   67.20   i******************************
   5     4.500   168    0.17   84.00   i****************************
   6     5.500   160    0.16  100.00   i***************************
   7     6.500     0    0.00  100.00   i
```

```
                    i------------------------------

ex = 3.500; vx = 2.917; sigma = 1.708
```

4.5 bdist and its sub-classes

the declaration of bdist is simply

```
dist class bdist; virtual : boolean procedure sample;;
```

it has an empty body, but makes the matching sample routine defined in its sub-classes available to a ref(bdist) variable. each sub-class of bdist contains parameters which specify the particular distribution, a boolean procedure sample which uses these parameters and the objects own procedure zyqsample. the initialising actions set type appropriately and check (where possible) for unlawful parameter values.

```
                bdist object : new bdist("bdist");

                   -------------------
                   !      bdist      !
                   -------------------
                   ! title   "bdist" !
                   ! virtual:report  !
                   ! obs           0 !
                   ! resetat     0.0 !
                   ! next       none !
                   ! join(r)         !
                   ! reset           !
                   ! writetrn        !   tab level
                   -------------------
                   ! u      59224073 !
                   ! ustart 59224073 !
                   ! type          0 !
                   ! antithetic false!
                   ! setseed         !
                   ! zyqsample       !
                   ! zyqfail         !
                   ! report          !   dist level
                   -------------------
                   ! virtual:sample  !   bdist level
                   -------------------
```

4.5.1 class draw

draw implements a probability distribution with chance a of returning true. if a <= 0.0, then sample always returns false; if a >= 1.0, then sample always returns true.

```
draw object : new draw("chance", 0.1);

                    -------------------
                    !      draw       !
                    -------------------
                    ! title  "chance" !
                    ! virtual:report  !
                    ! obs           0 !
                    ! resetat     0.0 !
                    ! next       none !
                    ! join(r)         !
                    ! reset           !
                    ! writetrn        !   tab level
                    -------------------
                    ! u      22046052 !
                    ! ustart 22046052 !
                    ! type          6 !
                    ! antithetic false!
                    ! setseed         !
                    ! zyqsample       !
                    ! zyqfail         !
                    ! report          !   dist level
                    -------------------
                    ! virtual:sample  !   bdist level
                    -------------------
                    ! a           0.1 !
                    ! sample          !   draw level
                    -------------------
```

outline

```
rdist class draw(a); real a;
begin
  boolean procedure sample;

actions:
  type := 6;
end***constant***;
```

actions

on generation, a draw object executes the actions of its prefixes: tab (which curtail title to 12 characters, if longer, and then call reset), dist (which set u and ustart, and enter the object into distq), and rdist (nil). the main class body actions set type to 6.

attributes

(for title, obs, resetat, next, join(r), reset, writetrn, see tab 2-4; for u, ustart, type, zyqsample, zyqfail, report, see dist, page 4-8:4-9).

real a represents the chance of true turning up.

boolean procedure sample (which matches the specification at the bdist level) returns true with probability a. if a <= 0.0 then sample returns false; if a >= 1.0 then sample returns true.

typical histogram

(1000 samples from a draw object with a = 0.40 histogrammed as 'if zyqsample then 1 else 2')

```
title       /   (re)set/   obs/  average/est.st.dv/  minimum/  maximum
chance            0.000   1000     1.599     0.490     1.000     2.000

cell/lower lim/    n/   freq/  cum %
                                       i-------------------------------
   0 -infinity     0    0.00    0.00   i
   1     0.500   401    0.40   40.10   i********************
   2     1.500   599    0.60  100.00   i******************************
   3     2.500     0    0.00  100.00   i
                                       i-------------------------------
```

4.6 procedure readdist

instead of generating distributions explicitly (and having to recompile should they need altering), distribution definitions may be read in from inf by such calls as

```
readdist(d, "arrival rate");
```

with the corresponding data in inf

```
arrival rate  negexp  0.2
```

this parallels the explicit

```
d :- new negexp("arrival rate", 0.2);
```

the routine is probably most used to input (long) empirical tables; and for experimentation with distributions before the model logic has been settled.

outline

```
procedure readdist(d, title); name d; value title;
                            integer d; text title;
begin
  check inf for title, if not found then error;
  check inf for type, if not recognised then error;
  d :- appropriate object with parameters read from inf;
end***readdist***;
```

since d is declared as ref(dist), it can be matched to any variable with qualification rdist, idist, or bdist. a simula error occurs if the qualification of d does not match th qualification of the generated object when the eventual assignment is made.

e.g. given ref(rdist)r;, the call

```
readdist(r, "die cast");
```

matched against the input

```
die cast  randint  1  6
```

causes a run time error as a randint object may not be assigned to an rdist variable.

typical valid inputs

given ref(rdist)r; ref(idist)i; ref(bdist)b;

```
         call              corresponding data from inf.

readdist(r, "name 1")     name 1  normal  10.0   5.0
readdist(r, "name 2")     name 2  uniform  5.0  10.0
readdist(r, "name 3")     name 3  erlang   0.0   1.0
readdist(r, "name 4")     name 4  constant 3.0
readdist(r, "name 5")     name 5  negexp   0.1
readdist(r, "name 6")     name 6  empirical  5
                                   1.0   0.0
                                   2.0   0.2
                                  15.0   0.4
                                  23.0   0.6
                                  24.0   1.0

readdist(i, "name 7")     name 7  poisson  3.0
readdist(i, "name 8")     name 8  randint    1     5

readdist(b, "name 9")     name 9  draw     0.5
```

chapter 5

class entity

entity objects model major simulation components. they may compete with one another for resources (res, bin), cooperate over stretches of time (wait/coopt), or even interrupt one another.

entity object : new entity("e");

```
 ----------------------
 !       entity       !
 ----------------------
 ! title        "e 1" !
 ! loop               !
 ! timein         0.0 !
 ! evtime         0.0 !
 ! priority         0 !
 ! cycle            0 !
 ! wants            0 !
 ! interrupted      0 !
 ! currentq      none !
 ! owner         none !
 ! terminated   false !
 ! ll            none !
 ! bl            none !
 ! rl            none !
 ! suc           none !
 ! pred          none !
 ! avail              !
 ! idle               !
 ! coopt              !
 ! interrupt(n)       !
 ! repeat             !
 ! into(q)            !
 ! out                !
 ! getserialno        !
 ! list               !
 ! insert             !
 ! insertaftercurrent !
 ! insertdelay0       !
 ! schedule(t)        !
 ! cancel             !   entity level
 ----------------------
```

outline

```
class entity(title); value title; text title; virtual: label loop;
begin
  real timein, evtime;
  integer priority, cycle, wants, interrupted;
  ref(queue)currentq;
  ref(entity)owner, ll, bl, rl, suc, pred;
  boolean terminated;

  boolean procedure idle;
  boolean procedure avail;
  procedure coopt;
  procedure interrupt(n); integer n;
  procedure repeat;
  procedure into(q); ref(queue)q;
  procedure out;
  procedure getserialno;
  procedure list;
  procedure insert;
  procedure insertaftercurrent;
  procedure insertdelay0;
  procedure schedule(t); real t;
  procedure cancel;

actions:
  if title.length > 10 then title :- title.sub(1,10);
  getserialno;
  evtime := -1.0;
  detach;
loop:;
  inner;
  terminated := true;
  if not idle then passivate;
end***entity***;
```

actions

the actions of the class body first curtail the length of the title to 10 characters if longer, and then append a 2-digit class serial number (by the call getserialno) so that individual objects can be distinguished (e.g. see object depicted on the previous page). evtime is set to -1.0 (a signal that the object is out of the event list) and then the object is detached. this gives the user the chance to individually name the entity before scheduling it. for example, compare

```
new entity("e").schedule(..);
```

with

```
e :- new entity("e"); e.schedule(..);
```

on return, the inner class body actions are executed (note the position of the virtual label loop). then terminated is set to true and the object is passivated (if in the event list). a demos error results if an attempt is made to schedule a terminated entity.

attributes

text title is a concatenation of the user given value (curtailed to 10 characters if longer) and a 2 digit class serial number.

label loop is by default situated before the inner statement. it can be redefined at inner levels. loop is used in conjunction with repeat to permit a textually neat description of a cyclic class body with repeated actions preceded by an initialisation. the rather ugly

```
initialisation;
while true do
begin
  repeated actions;
end;
```

can be replaced by

```
   initialisation;
loop:
   repeated actions;
   repeat;
```

real timein is set to time whenever an entity joins a queue. it is used to collect queue statistics on average wait times, etc.

real evtime holds the time at which an entity's next phase will be entered when it is in the event list. it is set to -1.0 when an entity is removed from the event list.

integer priority gives the entity's priority. this attribute is used by into whenever an entity joins a queue. an entity is ranked according its current value (larger values at the front end); tie breaks are resolved by placing later entries behind entities with the same priority but already in the queue. thus when an entity e enters a queue,

```
e.pred == none or e.pred.priority >= e.priority and
e.suc  == none or e.priority > e.suc.priority
```

integer cycle is incremented by 1 each time repeat is called. it thus gives a measure of how many times repeated entity actions have been executed.

integer wants is set to the amount currently requested when an entity is waiting upon a res or a bin, otherwise its value is zero. this enables release and give to operate a little more efficiently as, if the first entity in a resource queue wants more than is currently available, there is no point in trying to unblock it.

integer interrupted is normally zero, but is set to n by a call interrupt(n). it is the user's responsibility to clear it after dealing with an interrupt, e.g. by such coding as

```
if interrupted > 0 then
begin
  if interrupted = 1 then deal with interrupt of type 1 else
  if interrupted = 2 then deal with interrupt of type 2 else
  ..................;
  interrupted := 0;
end else no interrupt pending;
```

ref(queue)currentq is set to reference the queue an entity is currently a member of when it enters (set to q on a call into(q)). it is set to none when removed from a queue by a call on out. by queue we mean any queue or sub-class of queue, i.e. queue, res, bin, waitq, or condq.

ref(entity)owner references the entity if any which is currently coopting this entity, or none. it is set by a call on coopt and reset to none when the slave entity is next scheduled.

boolean terminated is set to true when the class body actions are exhausted (including those at inner levels). a terminated object may not be scheduled again (this is checked for in schedule).

ref(entity)ll, bl, rl reference neighbour entities in the event list, or none if the entity is not scheduled. the event list has been implemented as a leftist priority tree with special references root to the last entity in the tree, and current to the first entity in the tree (the one now operating).

the following invariant holds for a leftist priority tree t:

```
for all e belonging to t:
  e.ll == none => e.rl == none
  for all l belonging to e's left sub-tree
    l.evtime <= e.evtime
    for all r belonging to e's right sub-tree
      l.evtime < r.evtime < e.evtime

(e, l, r are all entities).
```

from which we may deduce that

```
e.ll.evtime = e.evtime => e.rl == none,
```

so that entities with equal event times form a linear sub-list connected by bl's and ll's, with the rl's of all but the first == none. this is handy to know when inserting delay 0 or after current.

ref(entity)suc, pred link together entities in queues. they are set to none when an entity leaves a queue. the suc of the last entity in a queue is none, as is the pred of the first.

boolean procedure avail; a call e.avail returns true if e is not currently coopted, i.e. if e.owner == none.

boolean procedure idle; a call e.idle returns true if e is currently out of the event list, i.e. e.evtime = -1.0.

procedure coopt; a call e.coopt causes a demos error if e.owner =/= none. otherwise, e.owner is set to current, and e is removed from its current queue if any.

procedure interrupt(n); integer n;. a call e.interrupt(n) removes an idle entity from its current queue, if any; and cancels a scheduled entity e. e.interrupted is set to n, and then e is scheduled delay 0 (at the current clock time, but as last entity scheduled for that time).

procedure repeat increments cycle by 1 and then goes to loop. (it is of course meant to be called locally: do not call e.repeat.)

procedure into(q); ref(queue)q; a call e.into(q) removes e from its current queue , if any. then it inserts e into q in priority order and sets e.currentq to q. into maintains the invariant

```
e.pred.priority >= e.priority > e.suc.priority
          (e.suc, e.pred =/= none).
```

into also maintains various statistics on q (average queue length, wait time, length now, etc.).

procedure out. e.out removes e from its current queue, if any. it also maintains queue statistics and sets e.suc, e.pred, and e.currentq to none.

procedure getserialno compares title with a list of names kept in zyqentnames. if a match is found in list member z, z.n is concatenated with title to give a text and a serial number. then z.n is incremented to reflect the next serial number. n is kept modulo 100. if no match is found, a new zyqenttitle object is entered at the head of zyqentnames with title as its text (and its n = 1). since getserialno is automatically called by the system, it should not be used explicitly again.

procedure list. a call, e.list, sends one line of information about an entity in the event list to outf. (it is used in snapsqs.)

```
e.evtime    in columns  6-15
e.title     in columns 17-28
```

```
e.ll.title  in columns 30-41
e.bl.title  in columns 43-54
e.rl.title  in columns 56-67
```

procedure insert is an auxiliary routine used by hold and schedule. it assumes that the correct evtime has been set locally and that ll, bl, rl == none. the routine starts from root and inserts in o(log n). special care has to be taken if the fresh entry is inserted as the new root, or as the new current. (root and current must be maintained; a new current must be resumed). the insert algorithm is:

1. e.evtime >= root.evtime. insert as the new root and quit.

2. set w :- root. from now on, w references the current level in the event list and e is to be inserted below w. l is short for w.ll; r is short for w.rl. we maintain the invariant w.evtime > e.evtime.

```
                             -----              -----
                             !   !              !   !
                             ! w !              ! e !
                             !   !              !   !
                             -----              -----
                            .     .
                           .       .
                     -----          -----
                     !   !          !   !
                     ! l !          ! r !
                     !   !          !   !
                     -----          -----

                     e.evtime < w.evtime
             l.evtime < r.evtime < w.evtime if r =/= none
             l.evtime <= w.evtime           if r ==  none

                     initial configuration
```

3. insert in the left sub-tree of w if we can (l.evtime <= w.evtime). let l :- w.ll;.

4. l == none? (implies r == none). insert e as new w.ll and quit. (a test is inserted for the new entry being current; and if so the new current e is resumed. however this case should not arise since it has been optimised out of calls on hold and schedule.)

```
                             -----
                             !   !
                             ! w !
                             !   !
                             -----
                            .
                           .
                     -----
                     !   !
                     ! e !
```

```
          !   !
          -----

e.ll == none, e.bl == w, e.rl == none; w.ll == e
```

5. (l =/= none and) e.evtime < l.evtime? go down one level (w :- w.ll;) and continue from step 3.

6. (l =/= none and) e.evtime = l.evtime? insert between l and w and quit.

```
                     -----
                     !   !
                     ! w !
                     !   !
                     -----
                    .     .
                   .       .
             -----           -----
             !   !           !   !
             ! e !           ! r !
             !   !           !   !
             -----           -----
            .
           .
     -----
     !   !
     ! l !
     !   !
     -----
```

e.ll == l, e.bl == w, e. rl == none; l.bl == e; w.ll == e
l.evtime = e.evtime < r.evtime < w.evtime

7. (l =/= none and) e.evtime > l.evtime? insert in the right sub-tree of w. let r :- w.rl;.

8. (l =/= none and) r == none? insert as the new w.rl and quit.

```
                     -----
                     !   !
                     ! w !
                     !   !
                     -----
                    .     .
                   .       .
             -----           -----
             !   !           !   !
             ! l !           ! e !
             !   !           !   !
             -----           -----
```

e.ll == none, e.bl == w, e.rl == none; w.rl == e
l.evtime < e.evtime < w.evtime

9. (l, r =/= none and) e.evtime < r.evtime? go down one level (w :- w.rl) and continue from step 3.

10. (l, r =/= none and) e.evtime >= r.evtime? insert between w and r and quit.

```
                             -----
                             !   !
                             ! w !
                             !   !
                             -----
                            .     .
                           .       .
                     -----           -----
                     !   !           !   !
                     ! l !           ! e !
                     !   !           !   !
                     -----           -----
                                    .
                                   .
                             -----
                             !   !
                             ! r !
                             !   !
                             -----
```

e.ll == r, e.bl == w, e.rl == none; w.rl == e; r.bl == e
l.evtime < r.evtime <= e.evtime < w.evtime

procedure insertaftercurrent is a fast o(1) insert used in demos scheduling routines such as acquire, take, ... e.insertaftercurrent sets e.evtime to time and then inserts e as the new bl of current. if current was root, root is set to reference the new entry e. no effect if evtime >= 0.0, i.e. the entity is already in the event list.

```
        -----           -----                              -----
        !   !           !   !                              !   !
        ! e !           ! w !                              ! w !
        !   !           !   !                              !   !
        -----           -----                              -----
                       .                                  .
                      .          =>                      .
                -----                              -----
                !   !                              !   !
                ! c !                              ! e !
                !   !                              !   !
                -----                              -----
               current                            .
                                                 .
                                           -----
                                           !   !
                                           ! c !
                                           !   !
                                           -----
```

```
c.bl==w;w.ll==e        e.ll==c,e.bl==w,e.rl==none
c.evtime <= w.evtime   c.evtime=e.evtime<=w.evtime
```

procedure insertdelay0; e.insertdelay0 inserts e at time, but as last entity scheduled for that time. the routine is a fast o(1) insert and starts from current following bl's until the insertion point is found. the exceptional case (where the new entry becomes root) is taken care of at the head of the routine. no effect if evtime >= 0.0, i.e. the entity is already in the event list.

procedure schedule(t); real t; e.schedule (t) schedules a passive entity e into the event list delay t, i.e. at time + t, t >= 0.0. an error results if e is terminated (e.terminated = true). the call has no effect if e is already scheduled (e.evtime >= 0.0. to reactivate such an e, you must first cancel it).

if t <= now, then e preempts current and the actions of e are taken up.

if now < t <= 0.0, e is scheduled delay 0.

if t >= 0.0, e.evtime is set and e is inserted from the top o(ln n) by a call on insert.

procedure cancel; a call e.cancel deletes the entity e from the event list. it has no effect if e is already passive (e.evtime < 0.0). a run time error results if an attempt is made to cancel the only entry in the event list (e == current == root). n.b. only the cases e == e.bl.ll are depicted; the cases e == e.bl.rl are very similar.

the routine sets evtime to -1.0 and then runs through the tests:

e.ll == none? (=> e.rl == none).

```
                -----                                  -----
                !   !                                  !   !
                ! w !                                  ! w !
                !   !                                  !   !
                -----                                  -----
               .     .                                .
              .       .                              .
        -----          -----                   -----
        !   !          !   !                   !   !
        ! e !          ! r !   =>              ! r !
        !   !          !   !                   !   !
        -----          -----                   -----
                      .                       .

                -----                    -----
                !   !                    !   !
                ! p !                    ! p !
                !   !                    !   !
```

-----   -----

if e is the root then a fatal error is incurred - an attempt to delete the last entity in the event list. otherwise there are two complications to overcome:

1) when we delete e we must remember to swing a non-empty right sub-tree of e.bl to the left.

2) if e == current, we must remember to locate and resume the new current. if e.bl.rl == none, this is e.bl; otherwise it is the leftmost entity in the right sub-tree of e.bl (shown as p).

the code outline is:

```
if this entity == root then error;
if this entity == current then
begin
  if bl.rl == none then current :- e.bl else
  begin
    locate the new current
    current :- e.bl.rl.ll.ll....ll;
    swing right sub-tree of e.bl in as e.bl.ll;
    e.bl.rl :- none;
  end;
  detach e from the event list;
  resume(current);
end else
begin
  swing right sub-tree of e.bl in as e.bl.ll;
  e.bl.rl :- none;
  detach e from the event list;
end;
```

e.rl == none? (and e.ll =/= none)

```
                -----                   -----
                !   !                   !   !
                ! w !                   ! w !
                !   !                   !   !
                -----                   -----
               .                       .
              .                       .
        -----                   -----
        !   !          =>       !   !
        ! e !                   ! l !
        !   !                   !   !
        -----                   -----
       .
      .
-----
!   !
```

```
!   !
! 1 !
!   !
-----
```

we merely have to set the links of e.ll and e.bl to bypass e and detach e from the event list. the only complication arises if e == root, when e.ll becomes the new root.

e.rl =/= none? (=> e.ll =/= none)

```
                      -----                                   -----
                      !   !                                   !   !
                      ! w !                                   ! w !
                      !   !                                   !   !
                      -----                                   -----
                     .                                       .
                    .                                       .
              -----                                   -----
              !   !                                   !   !
              ! e !              =>                   ! r !
              !   !                                   !   !
              -----                                   -----
             .     .                                 .
            .       .
      -----           -----                   -----
      !   !           !   !                   !   !
      ! l !           ! r !                   ! p !
      !   !           !   !                   !   !
      -----           -----                   -----
                     .                       .
                                            .
              -----                   -----
              !   !                   !   !
              ! p !                   ! l !
              !   !                   !   !
              -----                   -----
```

in this case, we swing the right sub-tree of e across in place of e, hanging the left sub-tree of e onto the leftmost entity, p,in the sub-tree r. if e == root then root is set to e.rl.

example: changing the event list algorithm

in operating systems simulations, it is sometimes necessary to interrupt current when a job of greater priority gets scheduled. how would we cope with this in demos?

one way, used to illustrate the ease with which parts of the standard can be ripped out and replaced, would be to change the basic event list algorithms where necessary. we have to change the tie break rule for entities in the event list from first-come, first served (fcfs) to priority and then fcfs for entities with equal event times and equal priorities. also it is now possible for a freshly scheduled entity to interrupt current. whenever an entity is scheduled at time, we have to check if it is the new current and resume it if it is.

neither insertaftercurrent nor now are now needed. we replace all calls on insertaftercurrent by calls on insertelay0 and change the value of the constant now to 0.0.

procedure insertdelay0 (page 5-9) may be coded as before with the while loop condition replaced by

```
 evtime > p.evtime or (evtime = p.evtime and priority >= p.priority)
```

as the new entry may be current, we also include as the last statement in the procedure body

```
if p == current then
begin
  current :- this entity;
  resume(current);
end;
```

in procedure hold (page 6-1), the check on current's successor in the event list becomes

```
 evtime > p.evtime or (evtime = p.evtime and priority >= p.priority)
```

procedure insert (pages 5-6:5-8) is also quite easy to program. all we need do is alter the two 'descend' tests labelled procedure insert1belowx and procedure insert2belowx to our old friend

```
 evtime > p.evtime or (evtime = p.evtime and priority >= p.priority)
```

and remove the call error(15,....).

procedure schedule (page 5-9) simplifies down a little. 'now' loses its significance - we can only interrupt current if we have greater priority. thus we replace the if-statement labelled preemptcurrent (yes, all if ... if ... else ... of it) by

```
if t < 0.0 then t := 0.0;
if t = 0.0 then insertdelay0 else
begin
  evtime := evtime + t;
  insert;
end;
```

5.2 class mainprogram

demos uses the same trick as class simulation to return control to the main program block (see common base]4, page 128[). demos creates an object of class mainprogram, referenced by ref(mainprogram)demos, to impersonate the main program block. its class body actions are

```
detach;
repeat;
```

every time this object becomes current, the detach statement causes the actions of the main program block to be taken up again; and every time hold is called from inside the main program block, the mainprogram object is rescheduled and the new current is resumed.

outline

```
entity class mainprogram;
begin
  detach;
  repeat;
end***entity***;
```

initialising actions

```
current :- root :- demos :- new mainprogram("demos");
demos.evtime    := 0.0;
```

this code installs demos as the first entity object in the event list at time 0.0. since demos is an entity it can be cancelled by calls on either passivate from within the main program block, or demos.cancel; and later rescheduled by a call demos.schedule(....). it can also seize resources. this may be useful when a simulation has a run length which is not known at the outset.

example of use

in a simulation model of a single ferry system, writing the main block as:

```
demos
  begin
    declarations;
```

```
  initialising statements;
  hold(480.0);
end;
```

will end the simulation run after 8 hours regardless of the state of the system. this is not realistic enough.

let the ferry stop working for the day when it returns to its home port, drops off its load and finds that it is not worthwhile starting another crossing, say time >= 465 minutes. an outline of the appropriate code is:

```
demos
  begin
    entity class ferry;
    begin
      load;
      cross and return;
      unload;
      if time < 465.0 then repeat;
      demos.schedule(0.0);
    end***ferry***;

    other declarations;
    initialising statements;
    passivate;
  end;
```

see also in the demos text book]1, example 5, pp. 67-71[.

for another example, consider the classic doctor's surgery which closes after 2 hours. any patients being consulted or waiting are seen, but not later arrivals.

let the doctor be modelled by ref(res)dr. we can code the demos block as:

```
demos
  begin
    ref(res)dr;
    other declarations;

    hold(540.0);      comment start at 9.00 o'clock;
    initialising statements;
    hold(120.0);
    dr.acquire(1);
  end;
```

chapter 6

global event list procedures

following the style of simula, there are two global scheduling procedures hold and passivate. (passivate is really redundant as cancel is local to entity in demos and serves the same purpose. but passivate is a little quicker; its base zyqpassivate is used in other demos routines.) there is also real procedure time which returns the value of current.evtime.

real procedure time simply returns current.evtime.

procedure zyqpassivate is equivalent to current.cancel, but is a little faster as it has a restricted environment. it is used in several demos routines to put current to sleep if it finds itself blocked. zyqpassivate does not give a trace. the algorithm is:

1. current == root? fatal error - an attempt to delete the only entry in the event list.

2. bl.rl == none? bl is the new current. goto 4).

3. (bl.rl =/= none). follow ll's of bl.rl to locate the new current. swing bl.rl over to bl.ll and set bl.rl => none.

4. delete the old current. resume the new current.

5. quit.

procedure passivate sends a one line trace to outf if the trace switch is on, and then calls zyqpassivate.

procedure hold(t); real t; delays current by t (>= 0.0). we first replace a negative t by 0.0. then we increment e.evtime by t. the following cases arise:

1. current == root? only one item in the event list. quit.

2. locate next entity p. if evtime < p.evtime, e is still current. quit.

3. current.evtime >= p.evtime. delete current and set current => p. insert the old current from the top. resume p. quit.

chapter 7

classes queue, waitq, and condq

7.1 class queue

tab class queue serves a dual purpose. primarily it is used in demos to prefix classes waitq, condq, res and bin. but queue is also usable in its own right as a convenient means of chaining several entities.

```
                 queue
                   !
                   !
         ---------------------
         !         !         !
         !         !         !
       condq     waitq    resource
                             !
                             !
                         ---------
                         !       !
                         !       !
                        res     bin
```

n.b. the only objects allowed in queues are entities.

an entity e is entered into a queue by a call e.into(q); and is removed from its (one) current queue by a call e.out. into and out maintain various queue statistics such as average queue time, queue length now, maximum queue length attained, etc.

```
                    queue object : new queue("cargo")

                      -------------------
                      !      queue      !
                      -------------------
                      ! title   "cargo" !
                      ! virtual:report  !
                      ! virtual:reset   !
                      ! obs           0 !
                      ! resetat     0.0 !
                      ! next       none !
                      ! join(r)         !
                      ! writetrn        !   tab level
                      -------------------
                      ! length        0 !
                      ! zeros         0 !
                      ! maxlength     0 !
                      ! qint        0.0 !
                      ! cum         0.0 !
                      ! lastqtime   0.0 !
                      ! first      none !
                      ! last       none !
                      ! list            !
                      ! report          !
                      ! reset           !   queue level
                      -------------------
```

outline

```
tab class queue;
begin
  integer length, zeros, maxlength;
  real qint, cum, lastqtime;
  ref(entity)first, last;

  procedure list;
  procedure report;
  procedure reset;

actions:
  if this queue is queue then join(queueq);
end***queue***;
```

actions

on generation, a queue object first executes the actions of its tab prefix (which curtail title to 12 characters, if longer, and then call reset). then the actions of the class body enter the object into queueq if it is a queue object (but not if it is in a sub-class; thus noqueue, waitq, condq, res, and bin objects are not entered into queueq).

attributes

(for title, obs, resetat, next, join, writetrn, see tab, page 2-4).

integer length is maintained by into and out (see entity, page 5-5) to reflect the current length of the queue, that is the number of entities currently waiting in the queue.

integer zeros is updated by 1 each time an entity leaves the queue after a 'zero' wait (zero is taken as < epsilon). it reflects the number of zero waits since clock time resetat.

integer maxlength holds the maximum queue length attained since clock time resetat.

real qint is used to record the time integral of the queue length since resetat. it is updated by calls on into and out, each of which increments qint by (e.timein-lastqtime)*length prior to entering or removing entity e into/from the queue.

real cum is used to maintain a time integral of the entity wait times (including zero waits). when an entity e leaves its current queue q (by an explicit or implicit e.out), cum is incremented by time-e.timein (and obs by 1).

real lastqtime records the last time at which an entity joined or left the queue; = resetat if obs = 0.

ref(entity)first references the first entity in the queue (one with the highest priority), or none if length = 0.

ref(entity)last references the last entity in the queue (one with the least priority), or none if length = 0.

procedure list writes to outf. it prints a heading followed by one line of information on each entity waiting in the queue. this line contains the entities position in the queue (columns 17-20), its title (in columns 22-33), its priority (in columns 35-42), and its time of entry (in columns 44-53). it is called from snapqueues.

```
                 entities waiting in l truck q
                 *****************************

               no object     priority entry in q

                1 l 8               4      9.805
                2 l 9               3      9.707
                3 l10               3      9.776
                4 l11               3      9.787
```

5	l12	3	9.790
6	l13	0	9.737
7	l14	0	9.760
8	l15	0	9.807

procedure report sends a report on the queue's status since resetat on one line to outf. it consists of:

its title in columns 1-12

its reset time in columns 14-23

the number of completed waits (calls on out) in columns 24-30

the maximum queue length since resetat in columns 31-36

the current queue length in columns 37-42

the average queue length (qint plus an end correction) / (time-resetat) in columns 44-53

the number of zero waits in columns 54-59

the average waiting time (time spent in the queue by entities which have now left) in columns 61-70. it includes zero waits.

n.b. columns 44-53 are skipped if time - resetat < epsilon; columns 61-70 are skipped if obs - 0.

```
title       /   (re)set/   obs/ qmax/ qnow/ q average/zeros/  av. wait
s truck q         0.000     97     4     3      0.796    25  8.082e-02
```

procedure reset sets zeros, obs, qint, cum to zero; lastqtime and resetat to time; maxlength to length. it does not affect length, first, or last.

example of use

cars arrive at a quay and are then carried across the water on a ferry. at the far side they are rescheduled and resume their actions again. a queue onboard is used to link cars together during the crossing. cars leave the ferry in reverse order. unloading and loading take 0.5; the ferry has a capacity of n cars and waits until it is full before sailing (summer season?).

```
ref(waitq)quay;

entity class car;
begin
queueforferry:
  quay.wait;
```

```
unload:
  drive away;
end***car***;
```

```
entity class ferry(n); integer n;
begin
  integer k; ref(entity)c; ref(queue)onboard;
  onboard :- new waitq("on board");
loop:
  for k := 1 step 1 until n do
  begin
    c :- quay.coopt;
    hold(load time);
    c.into(onboard);
  end;
  hold(crossing);
  for k := 1 step 1 until n do
    onboard.last.schedule(0.5*k);
  hold(0.5*n);
  return trip;
  repeat;
end***ferry***;
```

typical report (n = 2)

```
                              q u e u e s
                              ***********

title       /   (re)set/   obs/ qmax/ qnow/ q average/zeros/  av. wait
on board          5.000     90     2     2      0.947     0     10.259


                         w a i t   q u e u e s
                         *********************

title       /   (re)set/   obs/ qmax/ qnow/ q average/zeros/  av. wait
quay              0.000     92     1     0      0.137    80      1.486
quay        *     0.000     92    12    10      3.099    12     29.062
```

7.1.1 class noqueue

class noqueue is a trick used in the implementation of waitq where we really need two queues together - one for the masters, one for the slaves (but none for the little boy who lives down the lane). we declare

```
queue class noqueue;;
```

a noqueue object is not entered into the queue queueq by the class body actions at the queue level as these read

```
if this queue is queue then join(queueq);
```

noqueue objects generated by waitqs are entered by the class body actions of waitq into waitqq.

7.2 class waitq

waitq's are used in the master/slave synchronisation in which several entities cooperate together over a period of time. instead of having several entities moving down the event list together, it is simplest to single out one entity as the master and let it coopt the others for the period in question. slave entities are released from their coopting owner when next scheduled.

if the slaves are individually named, they may be coopted explicitly by using the routine coopt local to entity. however, this is not usually the case.

a waitq contains two queues: one for the slaves (its own queue level mechanism), and ref(queue)masterq for the masters.

```
                waitq object : new waitq("quay");

                  ------------------------
                  !       queue          !
                  ------------------------
                  ! title         "quay" !
                  ! virtual:report       !
                  ! virtual:reset        !
                  ! obs                0 !
                  ! resetat          0.0 !
                  ! next            none !
                  ! join(r)              !
                  ! writetrn             !   tab level
                  ------------------------
                  ! length             0 !
                  ! zeros              0 !
                  ! maxlength          0 !
                  ! qint             0.0 !
                  ! cum              0.0 !
                  ! lastqtime        0.0 !
                  ! first           none !
                  ! last            none !
                  ! list                 !
                  ! report               !
                  ! reset                !   queue level
                  ------------------------
                  ! masterq==new noqueue !
                  ! wait                 !
                  ! coopt                !
                  ! avail                !
                  ! find                 !   waitq level
                  ------------------------
```

outline

```
queue class waitq;
begin
  ref(queue)masterq;

  procedure wait;
  ref(entity)procedure coopt;
  boolean procedure avail(e,c); name e,c; ref(entity)e; boolean c;
  procedure find(e,c); name e,c; ref(entity)e; boolean c;

actions:
  masterq :- new noqueue(title);
  masterq.join(waitqq);
  join(waitqq);
end***waitq***;
```

actions

on generation, a waitq object first executes the actions of its tab prefix (which curtail title to 12 characters, if longer, and then call reset); and then those of its queue prefix (nil). then the main class body actions create a noqueue object and enter it into waitqq; then this waitq object is entered into waitqq. the net effect is that reports on waitqs occupy two lines: the masterq is reported on line 1, and this waitq (which gives 'slave' statistics) is reported on line 2. similarly, reset will reset both objects separately.

attributes

(for title, obs, resetat, next, join, writetrn, see tab, page 2-4; for length, zeros, maxlength, qint, cum, lastqtime, first, last, list, report, reset, see queue, page 7-2.)

procedure wait turns current into a (potential) slave resting in this queue.

```
q.wait;

     current.into(this queue);
     if q.masterq.first =/= none then
        masterq.first.insertdelay0;
     passivate;
```

the call places current in the slave queue in its priority order, activates the first master (if any) delay 0 (behind current), and then cancels current.

ref(entity)procedure coopt removes the first slave from the slave queue and coopts it on behalf of current. if the slave queue is empty, the requester is blocked in the master queue (in priority order) until it is the first entity and the slave queue is not empty.

```
e :- q.coopt;

     current.into(masterq);
     while length = 0 or current =/= masterq.first do
       passivate;
     current.out;
     if masterq.first =/= none and length = 0 then
        masterq.first.insertaftercurrent;
     p :- coopt :- first;
     p.coopt;
```

current is entered into masterq in priority order. if the length of the slave queue is zero or current is not the first entity in masterq, current is passivated and remains dormant until the above conditions are fulfilled. then current leaves the masterq and coopts the first slave entity (which will remove it from the slave queue). should it be possible to unblock the master queue, the first entity in masterq is activated after current.

boolean procedure avail(e, c); name e, c; ref(entity)e; boolean c; q.avail(e, condition) returns true if an entity e can be found in the slave queue of q satisfying the condition. if avail returns false, e is set to none. if avail returns true, e references the first entity in the slave queue satisfying the stated condition. notice that e and c are called by name; thus e may be assigned to and c is dynamically re-evaluated every time it is used.

procedure find(e, c); name e, c; ref(entity)e; boolean c; a call q.find(e, c) made by current delays current until q.avail(e, c) is true and then coopts e on behalf of current. if blocked, current is delayed in q.masterq.

```
q.find(e, c);

     current.into(masterq);
     while not q.avail(e, c) do
     begin
       p :- current.suc;
       if p =/= none then p.insertaftercurrent;
       passivate;
     end;
     p :- current.suc;
     current.out;
     e.coopt;
     if p =/= none then p.insertdelay0;
```

example of use

a cpu executes programs waiting in cpuq. each program has a cyclic life consisting of a burst on the cpu followed by an i/o transfer.

```
ref(waitq)cpuq;

entity class program;
begin
queueforcpuburst:
  cpuq.wait;
doio:
  hold(i/o transfer time);
  repeat;
end***program***;

entity class cpu;
begin
  ref(entity)e;
  e :- cpuq.coopt;
  hold(burst);
  e.schedule(0.0);
  repeat;
end***cpu***;
```

typical report

```
                         w a i t   q u e u e s
                         *********************

title       /   (re)set/   obs/ qmax/ qnow/ q average/zeros/  av. wait
cpu q             0.000    115     1     0      0.123    75      5.329
cpu q       *     0.000    115     2     0      0.730    41     31.732
```

7.3 class condq

condq's are used to implement waits until in demos. a simple waituntil algorithm has been chosen which suffices in most cases. its advantage is that its application is always easy to understand. if the condition is complicated then it will appear in the program as complicated. this accords with the notion that it should be easy to write down simple conditions, and possible to write down even the most complicated.

condq object : new condq("cond")

```
 -------------------
 !      condq      !
 -------------------
 ! title    "cond" !
 ! virtual:report  !
 ! virtual:reset   !
 ! obs           0 !
 ! resetat     0.0 !
 ! next       none !
 ! join(r)         !
 ! writetrn        !   tab level
 -------------------
 ! length        0 !
 ! zeros         0 !
 ! maxlength     0 !
 ! qint        0.0 !
 ! cum         0.0 !
 ! lastqtime   0.0 !
 ! first      none !
 ! last       none !
 ! list            !
 ! report          !
 ! reset           !   queue level
 -------------------
 ! all             !
 ! waituntil(c)    !
 ! signal          !   condq level
 -------------------
```

outline

```
queue class condq;
begin
  boolean all;

  procedure waituntil(c); name c; boolean c;
  procedure signal;
```

```
actions:
  join(condqq);
end***condq***;
```

actions

on generation, a condq object first executes the actions of its tab prefix (which curtail title to 12 characters, if longer, and then call reset), and then those of its queue prefix (nil). the main class body actions enter the condq object into the reportq condqq.

attributes

(for title, obs, resetat, next, join, writetrn, see tab, page 2-4; for length, zeros, maxlength, qint, cum, lastqtime, first, last, list, report, reset, see queue, page 7-2.)

boolean all is used as a flag in watuntil. if all is true, all entities waiting in the condq are activated to see if they can go when the condq is signalled. if all is false, only those at its head are tested and as soon as one condition evaluates to false, no more entities are tested. all can be used fruitfully when a condq contains entities waiting until on mixed conditions.

procedure waituntil(c); name c; boolean c; a call q.waituntil(c) enters current into the condq q. then c is evaluated. if c is true, then current leaves q at once and continues on. if c is false, current is passivated and remains in the condq until tested and c evaluates to true.

```
q.waituntil(c);

     current.into(this condq);
     while not c do
     begin
       if all then current.suc.insertaftercurrent;
       passivate;
     end;
     current.suc.insertaftercurrent;
     current.out;
```

procedure signal; a call q.signal activates the first entity in the condq q delay 0. current continues on uninterruptedly; when e becomes current, it evaluates its condition c again and leaves the condq if c evaluates to true. as it does so it promotes the next condq member (if any) into the event list immediately behind itself. in this way, the condq entities are peeled off and tested one by one. if all is true, all entities are tested; if all is false, only those at the head of the condq.

example of use

having been emitted by a furnace in a steel mill, billets are transported to a soaking pit area on bogies. there they are unloaded by a crane. if the pits are full the billets are dumped by the pit side and await a crane and a free pit. if a pit is free, they are loaded straight in.

```
ref(res)cranes, pits;
ref(condq)pitq;

entity class billet;
begin
  hold(transport time);
  cranes.acquire(1);
  if pits.avail = 0 then
  begin
    hold(dump by pit area time);
    cranes.release(1);
    pitq.waituntil(cranes.avail > 0 and pits.avail > 0);
    pits.acquire(1);
    cranes.acquire(1);
    hold(load from pit side time);
  end else
  begin
    hold(unload into pit time);
  end;
  cranes.release(1);
  pitq.signal;
  hold(soak time);
  ................
end***billet***;
```

typical report

```
                   c o n d i t i o n   q u e u e s
                   *******************************

title       /   (re)set/   obs/ qmax/ qnow/ q average/zeros/  av. wait
sq                0.000    114     1     0  5.038e-02    99  4.419e-03
```

chapter 8

class resource and its sub-classes

class resource is not meant to be used in its own right; it defines the common portion to the rather similar classes bin and res (this has not been pushed to extremes).

```
resource object : new resource("r", 4);

       -------------------
       !    resource     !
       -------------------
       ! title       "r" !
       ! virtual:report  !
       ! virtual:reset   !
       ! obs           0 !
       ! resetat     0.0 !
       ! next       none !
       ! join(r)         !
       ! writetrn        !   tab level
       -------------------
       ! length        0 !
       ! zeros         0 !
       ! maxlength     0 !
       ! qint        0.0 !
       ! cum         0.0 !
       ! lastqtime   0.0 !
       ! first      none !
       ! last       none !
       ! list            !   queue level
       -------------------
       ! avail         4 !
       ! extreme       4 !
       ! initial       4 !
       ! users         0 !
       ! sint        0.0 !
       ! lastrtime   0.0 !
       ! report          !
       ! reset           !   resource level
       -------------------
```

outline

```
tab class resource(avail); integer avail;
begin
  integer extreme, initial, users;
  real sint, lastrtime;

  procedure report;
  procedure reset;

actions:
  initial := avail;
end***resource***;
```

actions

on generation, a waitq object first executes the actions of its tab prefix (which curtail title to 12 characters, if longer, and then call reset); and then those of its queue prefix (nil). the main class body actions set initial to the actual parameter value of avail.

attributes

(for title, obs, resetat, next, join, writetrn, see tab, page 2-4; for length, zeros, maxlength, qint, cum, lastqtime, first, last, list, report, reset, see queue, page 7-2.)

integer avail reflects the currently available amount of the modelled resource. the invariants:

res: 0 <= avail <= initial

bin: 0 <= avail

are maintained by the acquire/release and take/give routines respectively.

integer extreme records the minimum value of avail attained since resetat for a res; and the maximum value of avail attained since resetat for a bin. extreme is maintained by calls on acquire and give respectively.

integer initial retains the initial value of the res or bin (for reporting purposes).

integer users records the number of completed calls on release (res) or give (bin). obs records the number of successful calls on acquire and take respectively.

real sint is used to maintain a measure of the resource usage since resetat. it holds the time integral of avail. in the case of a res object, this enables speedy computation of the res usage since resetat as a % of the maximum usage (time-resetat) * initial. in the case of a bin object, this enables computation of the average number of bin units available: remember that there is no upper limit here.

real lastrtime holds the time at which the last change to avail was made; or resetat if no changes have been made since.

procedure report (which is shared by res and bin) sends one line to outf consisting of:

title in columns 1-12

resetat in columns 14-23

users (number of completed calls on release if res, or give if bin) in columns 24-30

initial value of the resource in columns 31-35

extreme value of the resource in columns 36-40 (= maximum reached since resetat if res; minimum value reached since resetat if bin)

avail, the currently available number of resource units in columns 41-45

if res, the res usage (% of the maximum possible usage = (time-resetat) * initial) since resetat; if bin, the average number of units available since resetat; in columns 46-55

the average wait time of waiting entities since resetat in columns 56-65

the maximum queue length for waiting entities since resetat in columns 66-70

n.b. columns 46-55 are skipped if (time-resetat) < epsilon; columns 56-65 are skipped if obs = 0. notice also that certain portions of the queue statistics are not used in this report. but they can be made use of should you wish to write your own report routines.

procedure reset sets obs and zeros to zero; maxlength to length; lastqtime, lastrtime, and resetat to time; qint, sint, and cum to zero; and extreme to avail.

8.1 class res

res objects model the mutual exclusion synchronisation. entities acquire a portion of a res if sufficient of the res is available, and they have highest priority amongst entities waiting. otherwise, they wait passively in the blocked entity queue. when a resource (or portions of it) are returned, avail is updated and the blocked queue is tested to see who can go.

```
res object : new res("tugs", 2);

 -------------------
 !      res        !
 -------------------
 ! title    "tugs" !
 ! virtual:report  !
 ! virtual:reset   !
 ! obs           0 !
 ! resetat     0.0 !
 ! next       none !
 ! join(r)         !
 ! writetrn        !   tab level
 -------------------
 ! length        0 !
 ! zeros         0 !
 ! maxlength     0 !
 ! qint        0.0 !
 ! cum         0.0 !
 ! lastqtime   0.0 !
 ! first      none !
 ! last       none !
 ! list            !   queue level
 -------------------
 ! avail         2 !
 ! extreme       2 !
 ! initial       2 !
 ! users         0 !
 ! sint        0.0 !
 ! lastrtime   0.0 !
 ! report          !
 ! reset           !   resource level
 -------------------
 ! acquire(n)      !
 ! release(n)      !   res level
 -------------------
```

outline

```
resource class res;
begin
  procedure acquire(n); integer n;
  procedure release(n); integer n;

actions:
  if avail < 1 then error;
  into(resq);
end***res***;
```

actions

on generation, a res object first executes the actions of its tab prefix (which curtail title to 12 characters, if longer, then call reset); then those of its queue prefix (nil); and finally those of its resource prefix (which initialise initial to avail). the main class body actions check that avail is initialised to a sensible value (>= 1); if not a demos error is reported. then the res object is entered into resq.

attributes

(for title, obs, resetat, next, join, writetrn, see tab, page 2-4; for length, zeros, maxlength, qint, cum, lastqtime, first, last, list, report, reset, see queue, page 7-2; for avail, extreme, initial, users, sint, lastrtime, report, reset, see resource, page 8-2.)

procedure acquire(n); integer n; sees to it that a request for n units by an entity e is not granted until

e == first and avail >= n

the routine checks that n is sensible (1 <= n <= initial) giving a demos error if not. then current (== e) is entered into the res queue and passivated if it is not both first and its request can be granted. (this is different from a gpss capacity where only avail >= n is tested.) there it remains until tested (by a later call on release) and the above condition becomes true.

when tested and the condition is true, e leaves the blocked queue and does some book keeping on avail. then (this is only relevant for a delayed item) if the first blocked entity can now go, it is activated behind e. thus if an unblocking releases several entities they are peeled off one at a time, not interrupting current, but placed behind it in the event list.

```
acquire(n);

     current.into(this res);
     while current =/= first or n > avail do
       passivate;
     current.out;
     avail := avail - n;
     first.insertdelay0;
```

procedure release(n); integer n; checks to see that n is sensible (1 <= n <= initial - avail) and gives a demos error if not. (n.b. a full implementation could do better than this and check that n <= current's share of this res .) then avail is updated. if the first entity in the event list can now go, it is entered into the event list delay 0. in the intervening time it takes before it becomes current, its intended share is taken by some other entity, it will be passivated again.

```
release(n);

     avail := avail + n;
     first.insertaftercurrent;
```

example of use

customers enter a corner shop and look for a basket. if none can be found they quit in disgust. otherwise they take a basket and shop. when leaving, they queue to pay at a single checkout.

```
ref(count)disgusted;
ref(res)baskets, checkout;

entity class customer;
if baskets.avail = 0 then disgusted.update(1) else
begin
  baskets.acquire(1);
  shop;
  checkout.acquire(1);
  pay;
  checkout.release(1);
  baskets.release(1);
end***customer***;
```

typical report

c o u n t s

****

title /	(re)set/	obs
disgusted	8.000	4

r e s o u r c e s
****

title /	(re)set/	obs/	lim/	min/	now/	% usage/	av. wait/	qmax
baskets	8.000	25	20	0	1	81.347	0.000	1
check out	8.000	25	1	0	1	41.793	0.812	4

8.2 class bin

bin objects are used to model the producer consumer synchronisation. producers make items which are to be used by consumers. a consumer is blocked if no item is available when requested.

classes bin and res are rather similar but are worth implementing separately as bins have no upper limit. thus slightly different reports are in order, and since the same entity returns acquired resources to a res, a tight check can be made on res usage (not yet implemented).

```
                bin object : new bin("widgets", 4);

                   -------------------
                   !      bin        !
                   -------------------
                   ! title "widgets" !
                   ! virtual:report  !
                   ! virtual:reset   !
                   ! obs           0 !
                   ! resetat     0.0 !
                   ! next       none !
                   ! join(r)         !
                   ! writetrn        !   tab level
                   -------------------
                   ! length        0 !
                   ! zeros         0 !
                   ! maxlength     0 !
                   ! qint        0.0 !
                   ! cum         0.0 !
                   ! lastqtime   0.0 !
                   ! first      none !
                   ! last       none !
                   ! list            !   queue level
                   -------------------
                   ! avail         4 !
                   ! extreme       4 !
                   ! initial       4 !
                   ! users         0 !
                   ! sint        0.0 !
                   ! lastrtime   0.0 !
                   ! report          !
                   ! reset           !   resource level
                   -------------------
                   ! take(n)         !
                   ! give(n)         !   bin level
                   -------------------
```

outline

```
resource class bin;
begin
  procedure take(n); integer n;
  procedure give(n); integer n;

actions:
  if avail < 0 then error;
  into(binq);
end***bin***;
```

actions

on generation, a bin object first executes the actions of its tab prefix (which curtail title to 12 characters, if longer, then call reset); then those of its queue prefix (nil); and finally those of its resource prefix (which initialise initial to avail). the main class body actions check to see that avail >= 0 and give a demos entity if not. then the object is placed into binq.

attributes

(for title, obs, resetat, next, join, writetrn, see tab, page 2-4; for length, zeros, maxlength, qint, cum, lastqtime, first, last, list, report, reset, see queue, page 7-2; for avail, extreme, initial, users, sint, lastrtime, report, reset, see resource, page 8-2.)

procedure take(n); integer n; sees to it that a request for n units by an entity e is not granted until

e == first and avail >= n

the routine checks that n is sensible (n > 0) giving a demos error if not. then current (== e) is entered into the bin queue and passivated if it is not both first and its request can be granted. there it remains until tested (by a later call on give) and the above condition becomes true.

when tested and the condition is true, e leaves the blocked queue and does some book keeping on avail. then (this is only relevant for a delayed item) if the first blocked entity can now go, it is activated behind e. thus if an unblocking releases several entities they are peeled off one at a time, not interrupting current, but placed behind it in the event list.

```
take(n);

     current.into(this bin);
     while current =/= first or n > avail do
       passivate;
     current.out;
     avail := avail - n;
     first.insertdelay0;
```

procedure give(n); integer n; checks to see that m is sensible (n >= 0) and gives a demos error if not. then avail is updated. if the first entity in the event list can now go, it is entered into the event list delay 0. in the intervening time it takes before it becomes current, its intended share is taken by some other entity, it will be passivated again.

```
give(n);

     avail := avail + n;
     first.insertaftercurrent;
```

example of use

a small production line involves two processes: assembly and then packing. packers pack two previously assembled items per box. initially three items are assembled ready for packing (left over from the previous day.

```
ref(count)done;
ref(bin)assembled;

]assembled :- new bin("assembled", 3);
 done      :- new count("items packed");[

entity class assembler;
begin
  hold(assemble time);
  assembled.give(1);
  repeat;
end***assembler***;

entity class packer;
begin
  assembled.take(1);
  hold(pack time);
  done.update(1);
  repeat;
```

```
end***packer***;
```

typical report

```
                         c o u n t s
                         ***********

                title       /   (re)set/   obs
                done              0.000    167


                           b i n s
                           *******

title       /   (re)set/   obs/init/ max/ now/ av. free/ av. wait/qmax
assembled         0.000    166    2    4    0     1.083 5.344e-02    1
```

chapter 9

odds and ends

9.1 some primitive reporting aids

the demos report and trace routines repeatedly use certain more primitive routines. these are:

procedure clocktime which sends the current clocktime (centred) on a fresh line to outf.

procedure box(t); value t; text t; which sends the text value t centred inside a rectangular frame of asterisks (5 rows by 70 columns) and then two blank lines to outf.

procedure printreal(x); real x; which sends to outf a field of width 10 characters containing the real value x. fixed point (3 decimal places) where possible; otherwise floating point. if the fixed point representation admits only 2 significant figures x is printed floating point.

procedure edit(t, n); value t; text t; integer n; concatenates t with blanks(2).putint(n), where n = abs(n) modulo 100. if t.length > 10, then t is first subbed down to length 10.

9.2 tracing

demos traces are sent to outf. tracing in demos is controlled by an integer flag zyqtrace, initially zero. every time a resource is requested, acquired, every time an entity holds, etc., etc., code inside the controlling routine tests zyqtrace. if it is positive, a one line print out of the current event is sent to outf, e.g.

```
1.237 furnace 1    acquires 3 of power
```

if zyqtrace is zero, the printout is skipped.

zyqtrace is incremented by each call on the global routine trace; and is decremented (if positive) by each call on the global routine notrace.

outline

```
procedure note(index, action, e, l, t1, n);
    value action; text action; integer index, n;
    real t1; ref(entity)e; ref(tab)l;
begin
  print time in columns 1-10;
  print current.title in columns 12-23 (left justified);
  print action in columns 25-35;
branchforrestofline:
  case index of
  begin
  1:   e.coopt;
  2:   e.schedule(t);
  3:   current terminates;
  4:   res.acquire(n)  - blocked;
  5:   res.acquire(n)  - seizes;
  6:   res.release(n);
  7:   bin.take(n)     - blocked;
  8:   bin.take(n)     - seizes;
  9:   bin.give(n);
  10-13:  not in use;
  14:  q.wait          - blocked;
  15:  q.find          - blocked;
  16:  q.find          - finds;
  17:  q.coopt         - blocked;
  18:  q.coopt         - coopts;
  19:  q.waituntil     - blocked;
  20:  q.waituntil     - leaves;
  21:  hold(t);
  22:  passivate;
  23:  e.cancel;
  24:  e.interrupt(n);
  25:  q.signal;
```

```
  end***case***;
  outf.outimage;
end***note***;
```

```
procedure trace;
begin
  if zyqtrace = 0 then print boxed heading - trace starts;
  zyqtrace := zyqtrace + 1;
end***trace***;

procedure notrace;
begin
  zyqtrace := zyqtrace - 1;
  if zyqtrace = 0 then print boxed heading - trace off;
end***notrace***;

integer zyqtrace;
real zyqnotelastt;
ref(entity)zyqnotelaste;
```

trace related quantities

integer zyqtrace is the controlling flag for event tracing. it is incremented by trace and decremented by notrace (when positive). each call on note is of the form

```
if zyqtrace > 0 then note(......);
```

real zyqnotelastt holds -15.0 (arbitrary choice) if tracing is off; otherwise it holds the clock time at which the last trace line was printed. the time columns of the trace line (1-10) are skipped if the time of this note is the same as the time of the last note (to 3 decimal places).

ref(entity)zyqnotelaste references none if tracing is off; otherwise the perpetrator of the last event. if zyqnotelaste == current, columns 12-23 of the trace line are skipped.

procedure trace increments zyqtrace by 1, and prints a boxed message stating that tracing commences if zyqtrace is now 1.

procedure notrace has no effect if zyqtrace is negative. otherwise, zyqtrace is decremented by 1. if zyqtrace is now zero, a boxed message is printed stating that tracing has been switched off.

procedure note sends one line of trace output to outf.

typical trace (switched on at time = 0.35, and off at 0.40)

```
                         clock time =      0.350
*********************************************************************
*                                                                   *
*                 t r a c i n g   c o m m e n c e s                 *
*                                                                   *
*********************************************************************

      time/ current     and its action(s)

     0.350 demos        holds for  0.050, until 0.400
     0.354 l 8          schedules l 9 at 0.400
                        signals sq
                        waits in l truck q
     0.361 s digger 1   schedules s 4 now
                        coopts l 8 from l truck q
                        holds for  0.333, until 0.694
           s 4          ***terminates
     0.370 l digger 2   schedules l 6 now
                        interrupts s digger 1, with n = 2
                        cancels s digger 1
                        waits in l truck q
           l 6          ***terminates
           s digger 1   schedules l 8 now
                        w'until in sq
           l 8          signals sq
                        waits in l truck q
           l digger 2   coopts l 8 from l truck q
                        holds for  0.081, until 0.451
     0.376 s 5          schedules s 6 at 0.452
                        signals sq
                        waits in s truck q
           s digger 1   leaves sq
                        coopts s 5 from s truck q
                        holds for  0.083, until 0.459


                         clock time =      0.400
*********************************************************************
*                                                                   *
*              t r a c i n g   s w i t c h e d   o f f              *
*                                                                   *
*********************************************************************
```

9.3 procedure error

all errors and warnings generated by demos are recorded on sysout. n.b. this includes errors and warnings from dist objects generation and calls on readdist too (see chapter 4) as well as from the procedure error herein outlined.

outline

```
procedure error(no, e, q, n, call); value call; text call;
                   integer no, n; ref(entity)e; ref(tab)q;
begin
  print clock time;
  print "error in demos program";
  print cause of error (given by parameter call);
  print who is current;
  case n of
  begin
  e1:  e.coopt;
  w2:  e.into(q);         - q == none.
  w3:  e.into(q);         - e is already in a queue.
  e4:  r.acquire(n);      - n < 0.
  e5:  r.acquire(n);      - n > r.limit.
  e6:  b.take(n);         - n < 0.
  e7:  r.release(n);      - n < 0.
  e8:  r.release(n);      - n > r.limit.
  e9:  b.give(n);         - n < 0.
  e10: new res(title, l); - l < 1.
  e11: new bin(title, l); - l < 1.
  w12: x.cancel;          - x is idle.
  e13: x.cancel;          - x==current==root, i.e. x only entity.
  e14: e.schedule(t);     - e is terminated.
  e15: ****implementation error****;
  w16: e.schedule(t);     - e already scheduled.
  w17: e.cancel;          - e is idle.
  w18: q.join(r);         - r == none.
  w19: new histo(t,l,u,n);- l > u.
  e20: new empirical(t,s);- s = 1.
  end;
  if fatal then
  begin
    close inf, outf if redirected;
    print "deliberately induced rte";
    n := 0;
    n := 1/n;
  end;
end***error***;
```

case labels marked e induce fatal errors; case labels marked w induce warnings from which recoveries are made.

examples of error messages

given ref(res)r and the initialisation

```
r :- new res("harbour", 5);
```

a subsequent call 'r.acquire(6);' would generate something like

```
----------------------------------------------------------------------
----------------------------serious  error----------------------------
----------------------------------------------------------------------
**cause : call on 'r.acquire(n); ref(res)r; integer n;'.
        current == boat 8
        n = 6
        request for too many units from res 'harbour'.
        set 0 < n <= r.limit (= 5)
----------------------------------------------------------------------
---------------------------program  aborted---------------------------
----------------------------------------------------------------------
```

as a second example, consider an attempt to cancel an idle entity. this does not cause the program to stop but does generate a warning on sysout.

```
----------------------------------------------------------------------
----------------------------serious  error----------------------------
----------------------------------------------------------------------
**cause : call on 'e.cancel; ref(entity)e;'.
        current == battleship18
        attempt to cancel non-scheduled entity e == 'sub 4'.
        statement ignored.
```

9.4 global report and reset

all reports are sent to outf. report is called automatically at the end of each demos program unless switched off. report can be switched off by a call on the global routine noreport.

should you wish to report only selected facilities, say the res and bin objects only, you may do so by such coding as

```
ref(reportq)r;

for r :- resq, binq do
  r.report;
```

similarly for reset.

procedure reset is a global routine which cycles down the demos reportqs and calls the reset routine local to every user-generated demos facility therein.

outline

```
boolean zyqreport; (= true initially)

procedure noreport;
  zyqreport := true;

procedure report;
begin
  ref(reportq)r;
  clocktime;
  box("r e p o r t");
  for r :- distq, empq, accumq, countq, tallyq, histoq,
           resq,  binq, queueq, condq,  waitq do
  begin
    if r =/= none then
    begin
      2 newlines;
      r.report;
    end;
  end;
end***report***;

procedure reset;
begin
  ref(reportq)r;
  for r :- distq, empq, accumq, countq, tallyq, histoq,
           resq,  binq, queueq, condq,  waitq do
  begin
```

```
    r.reset;
  end;
end***reset***;
```

report and reset related quantities

boolean zyqreport controls the final simulation report. if it is true, then such a report is issued; if it is false (set by a call on noreport) then the automatic final report is skipped.

procedure noreport sets zyqreport to false.

procedure report is a global procedure which may be called at any time and sends a report on the current status of all user-created demos facilities to outf.

procedure reset resets each user-created demos facility.

example of use

suppose a simulation is to last for 1000 time units, takes 100 units to settle down from a cold start and we wish reports every 300 time units thereafter. instead of a hold(1000.0) in the demos block, we put

```
hold(100.0);
reset;
for k := 1 step 1 until 3 do                 ]integer k;[
begin
  report;
  reset;
end;
noreport;
```

the call on noreport stops the now unwanted automatic report which would duplicate (more or less - they are separated by a hold(0.0)) the third explicit report call.

9.5 queue and event list snapping

both these routines send snapshots to outf. snapqueues lists out all entities currently waiting in user generated queues, waitqs, and condqs (in that order). snapsqs traverses the event list and prints out the scheduled entities in order.

outline

```
proceure snapqueues;
begin
  ref(reportq)r;
  clocktime;
  box("list of all passive objects");
  for q :- queueq.first, condq.first, waitq.first do
  begin
    if q =/= none then
    begin
      q.list;
      q :- q.next;
    end;
  end;
end***snapqueues***;

procedure snapsqs;
begin
  clocktime;
  box("event list");
  traverse the event list from current to root and
  print on one line for each entity e:
     e's ordinal number in the event list in columns 1-5;
     e's event time in columns 6-15;
     e's title  in columns 18-29;
     e.ll.title in columns 31-42;
     e.bl.title in columns 45-56;
     e.rl.title in columns 59-70;
end***snapsqs***;
```

typical calls

a call on snapqueues:

```
                       clock time =     10.000
**********************************************************************
*                                                                    *
*           l i s t   o f   p a s s i v e   o b j e c t s            *
*                                                                    *
**********************************************************************


                      entities waiting in s truck q
                      *****************************

                    no object     priority entry in q

                     1 s98               0      9.940
                     2 s99               0      9.943
                     3 s 0               0      9.993


                      entities waiting in l truck q
                      *****************************

                    no object     priority entry in q

                      **** the queue is empty. ****
```

a call on snapsqs:

```
                      clock time =     10.000
**********************************************************************
*                                                                    *
*                        e v e n t   l i s t                         *
*                                                                    *
**********************************************************************


    n/  ev.time/obj. title / ll           bl            rl

    1    10.000 demos                     l16
    2    10.033 l digger 1                l16
    3    10.038 l16          demos        l digger 2    l digger 1
    4    10.038 l digger 2   l16          s digger 1
    5    10.046 s digger 1   l digger 2   s 1
    6    10.064 s 1          s digger 1
```

9.6 epsilon - the grain of time

real epsilon, set to 0.00001 by the actions of the demos prefix, is used in several routines to represent the smallest grain of time, e.g.

```
if time - resetat < epsilon then count as zero else update;
```

in simulations with very small or very large time scales, a different grain of time may be more appropriate. epsilon may be changed directly by assignment, for example

```
epsilon := 0.00000005
```

chapter 10

the prefix demos

the demos prefix contains the declarations of all the classes, procedures, variables documented in chapters 2 through 9. in addition, it contains certain initialising actions (setting up the event list, creating the standard reportq's, etc) and finalising actions (closing the files inf and outf perhaps, and issuing the final report).

outline

```
class demos;
begin
                comment data collection devices;

  class tab(title); value title; text title;
           virtual: procedure reset, report;

  tab class count;
  tab class tally;
  tab class accumulate;
  tab class histogram;
  tab class regression;
  tally class notally;


             comment the random number generators;

  integer procedure zyqnextseed;
  procedure setseed(n); integer n;

  tab class dist;
  class reportq;

  dist  class rdist;
  rdist class constant;
  rdist class empirical;
  rdist class erlang;
  rdist class negexp;
  rdist class normal;
  rdist class uniform;
```

```
dist  class idist;
idist class poisson;
idist class randint;

dist  class bdist;
bdist class draw;

procedure readdist;


                  comment reporting aids;

procedure clocktime;
procedure box(t); value t; text t;
text procedure edit(t, k); value t; text t; integer k;
procedure printreal;


         comment the central declaration - entity;

class entity;
entity class mainprogram;
class zyqenttitle;


                comment event list routines;

real procedure time;
procedure zyqpassivate;
procedure passivate;
procedure hold(t); real t;


             comment queue and its subclasses;

tab   class queue;
queue class noqueue;

queue class resource;
resource class res;
resource class bin;

queue class waitq;
queue class condq;


               comment the tracing routines;

real zyqnotelastt;
ref(entity)zyqnotelaste;
integer zyqtrace;
```

```
procedure trace;
procedure notrace;
procedure note(index, action, e, l, t1, n);
     value action; text action; integer index, n;
```

```
          real t1; ref(entity)e; ref(tab)l;


     procedure error(no, e, q, n, call); value call; integer no, n;
          text call; ref(entity)e; ref(tab)q;
     procedure abort;


                    comment the snapping routines;

     procedure report;
     procedure noreport;
     procedure reset;
     procedure snapqueues;
     procedure snapsqs;


                       comment local variables;

     ref(reportq)empq, tallyq, accumq, histoq, countq, regressq,
                 distq, resq, binq, queueq, condqq, waitqq;
     text tallyheading, accumheading, distheading;
     text headingrtn, stars, minuses, zyqreason, zyqrecvry;
     text resheading, binheading, qheading;
     text array disttype(0:9);
     integer zyqseed, zyqmodulo;
     ref(infile)inf;
     ref(outfile)outf;
     ref(zyqenttitle)zyqentnames;
     ref(mainprogram)demos;
     ref(entity)root, current, zyqp;
     real now, simperiod, epsilon;
     boolean zyqreport;

     zyqreport   := true;
     epsilon     := 0.00001;
     headingrtn  :-copy("title       /   (re)set/   obs");
     accumheading:-copy("/  average/est.st.dv/  minimum/  maximum");
     distheading :-copy("/type     /        a/        b/     seed");
     tallyheading:-accumheading;
     resheading  :-copy("/ lim/ min/ now/  % usage/ av. wait/qmax");
     binheading  :-copy("/init/ max/ now/ av. free/ av. wait/qmax");
     qheading    :-copy("/ qmax/ qnow/ q average/zeros/  av. wait");

     disttype(0) :-copy("undefined");
     disttype(1) :-copy("normal");   disttype(2) :-copy("uniform");
     disttype(3) :-copy("erlang");   disttype(4) :-copy("randint");
     disttype(5) :-copy("negexp");   disttype(6) :-copy("poisson");
     disttype(7) :-copy("draw");     disttype(8) :-copy("constant");
     disttype(9) :-copy("empirical");
```

```
accumq :- new reportq("a c c u m u l a t e s",
                          headingrtn, accumheading);
regressq:-new reportq("r e g r e s s i o n s", notext, notext);
countq :- new reportq("c o u n t s", blanks(20), headingrtn);
distq  :- new reportq("d i s t r i b u t i o n s",
                          headingrtn, distheading );
empq   :- new reportq("e m p i r i c a l s", notext, notext);
histoq :- new reportq("h i s t o g r a m s", notext, notext);
tallyq :- new reportq("t a l l i e s",headingrtn,tallyheading);
resq   :- new reportq("r e s o u r c e s",headingrtn,resheading);
binq   :- new reportq("b i n s",headingrtn,binheading);
queueq :- new reportq("q u e u e s",headingrtn,qheading);
condqq :- new reportq("c o n d i t i o n   q u e u e s",
                          headingrtn, qheading);
waitqq :- new reportq("w a i t   q u e u e s",
                          headingrtn, qheading);

stars :- blanks(70);
while stars.more do
  stars.putchar('*');

minuses :- blanks(70);
while minuses.more do
  minuses.putchar('-');

inf          :- sysin;
outf         :- sysout;
zyqreason    :- copy("**reason      : ");
zyqrecvry    :- copy("**recovery    : ");
zyqmodulo    := 67099547;
zyqseed      := 907;
now          := -10&20;
zyqnotelastt := -15.0;

             comment initialise the event list;

current :- root  :- demos :- new mainprogram("demos");
demos.evtime := 0.0;
demos.title  :- demos.title.sub(1, 5);

inner;

     comment program executed - now tidy up and report;

zyqp :- demos.nextev;
while (if zyqp == none then false else zyqp.evtime = time) do
begin
  hold(0.0);
  zyqp :- demos.nextev;
end;
```

```
   if zyqreport then report;
    if inf =/= sysin  and inf.image  =/= notext then inf.close;
   if outf =/= sysout and outf.image =/= notext then outf.close;
end***demos***;
```

actions

the actions of the demos prefix establish two texts - stars a row of asterisks and minuses a row of minus signs. they are used in the various reports and error messages. inf and outf are set to their default values, inf == sysin and outf == sysout. two subsidiary texts, zyqreason and zyqrecvry are then created - they are used in zyqfail (see chapter 4). zyqmodulo is set to 67099547 (the base for our random number routines) and the start value for the well spread seeds, zyqseed, is set to 907. now is initialised to a large negative value, and zyqnotelastt to an 'impossible' value for time.

then the event list is set up. the impersonating entity demos is installed as the first (current) and last (root) entity in the event list at time 0.0. (since demos.title is set by its class actions to "demos 1", it is set back again to just "demos".

on meeting inner; the user written program is executed. on return, the while loop ensures that all actions scheduled for the current clock time are executed. note that if these actions cause further actions to be executed at this time (by releasing resources os signalling a condq, for example) then these subordinate actions are also allowed to complete (recursive). a hold(0.0); is thus not good enough. then a report is automatically issued if not switched off and finally, inf and outf are closed if they are open.

attributes

ref(reportq)empq is the standard reportq for holding empirical objects.

ref(reportq)tallyq is the standard reportq for holding tally objects.

ref(reportq)accumq is the standard reportq for holding accumulate objects.

ref(reportq)histoq is the standard reportq for holding histogram objects.

ref(reportq)regressq is the standard reportq for holding regression objects.

ref(reportq)countq is the standard reportq for holding count objects.

ref(reportq)distq is the standard reportq for holding dist objects; i.e. objects of the classes erlang, constant, normal, negexp, uniform, randint, poisson, draw.

ref(reportq)resq is the standard reportq for holding res objects.

ref(reportq)binq is the standard reportq for holding bin objects.

ref(reportq)queueq is the standard reportq for holding queue objects.

ref(reportq)condqq is the standard reportq for holding condq objects.

ref(reportq)waitqq is the standard reportq for holding waitq objects.

text headingrtn holds the common portion of most of the report headings, namely

"title / (re)set/ obs"

it is used in the routine writetrn local to class tab.

text accumheading is used in reports on accumulate objects. the heading for these reports consists of a single line, first headingrtn (columns 1 -30) and then accumheading (columns 31 - 70). the text value is:

"/ average/est.st.dev./ minimum/ maximum"

the text object is also referenced by tallyheading (i.e. if you change one, you change them both).

text tallyheading is used in reports on tallys. the heading for these reports consists of a single line, first headingrtn (columns 1 - 30) and then tallyheading (columns 31 - 70). tallyheading references he same text object as accumheading.

text distheading is used in reports on dists (but not empiricals). the heading for these reports consists of a single line, first headingrtn (columns 1 - 30) and then distheading (columns 31 - 70). the text value is:

"/type / a/ b/ seed"

text resheading is used in reports on ress. the heading for these reports consists of a single line, first headingrtn (columns 1 - 30) and then resheading (columns 31 - 70). the text value is:

"/ lim/ min/ now/ % usage/ av. wait/qmax"

text qheading is used in reports on queues (all types: queue, waitq, condq). the heading for these reports consists of a single line, first headingrtn (columns 1 - 30) and then qheading (columns 31 - 70). the text value is:

```
"/ qmax/ qnow/ q average/zeros/  av. wait"
```

text reason is used in the dist failure routines.

text zyqrecvry is used in the routine zyqfail local to class empirical.

text stars is a text of length 70 asterisks. it is used frequently in reports and snapping routines (often subbed).

text minuses is a text of length 70 minus signs. it is used in the error routines (to make their headings and tails stand out), and also in reports should any fields be null (no readings made, for example).

text array disttype(0:9) is used in dist reports and error messages. each dist subclass is given its own identifying type. using this as index into disttype is a cheap way of returning the class identifier. disttype is initialised by the actions of the demos prefix to:

```
disttype(0) = "undefined"
disttype(1) = "normal"
disttype(2) = "uniform"
disttype(3) = "erlang"
disttype(4) = "randint"
disttype(5) = "negexp"
disttype(6) = "poisson"
disttype(7) = "draw"
disttype(8) = "constant"
disttype(9) = "empirical"
```

remember to extend disttype should you add in your own extra sub-classes to dist.

integer zyqseed is the base for the well spread seeds. it is set to 907.

integer zyqmodulo is the divisor in the demos random number generator (see chapter 4). it is initialised to 67099547.

ref(infile)inf is the input file for reading in distributions. it is initialised to == sysin and automatically closed by demos. if you wish to read in integers, reals, etc. from inf then do so by inf.inint, inf.inreal, etc. or use inspect. see also chapter 11.

ref(outfile)outf is the output file for traces and reports, but not errrors (which go to sysout). outf is initialised to sysout, but may be reassigned by the user. it is closed automatically by demos. see also chapter 11.

ref(zyqenttitle)zyqentnames is a behind the scenes reference to a list of entity names valid for this program. each time an entity object is generated, its title is compared with the list of names in this list. if found, title is concatenated with the appropriate serial number for this class, if not found, a fresh entry is made into the list, and title is concatenated with " 1".

ref(mainprogram)demos is a reference to the prefix block impersonating entity. see further in chapter 5.2.

ref(entity)root is a reference to the last entity in the event list. it is initialised to demos (see immediately above) and thereafter maintained by the event list routines.

ref(entity)current is a reference to the first entity in the event list. it is initialised to demos (see above) and thereafter maintained by the event list routines.

ref(entity)zyqp is used by the system at the end of a simulation run to locate demos.nextev. if its evtime = time, then demos is held for 0.0, thus ensuring that all events that should be carried out at the final clock time, are caried out.

real now is a constant set to -10&20. it is used in scheduling when one wishes to preempt current instead of entering the event list delay 0 (write schedule(now)).

real epsilon is the finest grain of time recognised by the report routines. for example, an entity which leaves a queue less than epsilon after having joined it, is counted as a zero. it may be reset from its initial value of 0.00001. see also chapter 9.6.

boolean zyqreport is the switch, initially true, which decides whether or not a final report should automatically be given. see chapter 9.5.

chapter 11

redirecting input and output in demos

the only input routine defined in demos is readdist which reads dist descriptions from inf. the many output routines (report, trace) output sequentially to outf (warnings and errors go straight to sysout). initially, inf == sysin and outf == sysout. inf and outf may be dynamically reassigned by the programmer as often as is required. the best demos can do is to automatically close inf and outf at program end should they still be open. (this includes when the program has been deliberately aborted due to demos detected errors.) but if several such files have been created, you should explicitly close all but the last yourself. similarly for a sequence of infiles.

```
class demos;
begin
  comment***definition of i/o files***;
  ref(infile)inf;
  ref(outfile)outf;
  inf  :- sysin;
  outf :- sysout;
  inner;           (program is executed)
closefiles:
   if inf =/= sysin  and inf.image  =/= notext then
inf.close;
  if outf =/= sysout and outf.image =/= notext then
outf.close;
end***demos***;
```

examples of use

reading from sysin:

```
demos
  begin
    integer n;
    ref(idist)r;

    n := inint;
    readdist(r, "r");
    .................
  end;
```

reading from inf:

```
demos
  begin
    integer n;
    ref(rdist)r;
    inf :- new infile("data");
    inf.open(blanks(80));
    ...............
    n := inf.inint;
    readdist(r, "r");
end;
```

or

```
demos
  begin
    integer n;
    ref(rdist)r;
    inf :- new infile("data");
    inf.open(blanks(80));
    .............
    inspect inf do
    begin
      n := inint;
      readdist(r, "r");
    end;
  end;
```

keeping the report on sysout and diverting the trace to another file:

```
demos
  begin
    .............
    outf :- new outfile("trace file");
    outf.open(blanks(70));
    trace;
    start the simulation off;
    hold(simulation period);
    outf.close;
    outf :- sysout;
  end
```

writing the trace to one file, the report to another:

```
demos
  begin
    .............
    outf :- new outfile("trace file");
    outf.open(blanks(70));
    trace;
    start the simulation off;
    hold(simulation period);
```

```
  outf.close;
  outf :- new outfile("report file");
  outf.open(blanks(70));
end
```

appendix r

r e f e r e n c e s

1) g. m. birtwistle "demos - discrete event modelling on simula", macmillan, 1979.

2) g. m. birtwistle, o. j. dahl, b. myrhaug, and k. nygaard "simula begin", studentlitteratur, 1973.

3) a. clementson "ecsl manual", university of birmingham, england, 1973.

4) o. j. dahl and k. nygaard "simula 67 common base language", norwegian computing centre, oslo, norway, 2nd edition, 1970.

5) d. y. downham and f. d. k. roberts "multiplicative congruential pseudo-random number generators", computer journal, 10(1), 1967, pp. 74-77.

6) a. t. fuller "the period of pseudo-random numbers generated by lehmer's congruential method", computer journal, 19(2), 1976, pp. 173-177.

7) m. ohlin "next random - a method of fast access to any number in the random generator cycle", simula newsletter, 6(2), 1977, pp. 18-20.

8) j. vaucher "simulation data structures using simula 67", proc. winter simulation conference, 1971, pp. 255-260.

appendix a

l i s t i n g   o f   d e m o s

```
   class demos;
b1 begin
     comment----d a t a   c o l l e c t i o n   d e v i c e s----
     *
     * this level contains the data collecting mechanisms
     * and their printing routines. the definitions are:
     *
     *      accumulate  count  histogram  tally
     *
     * accumulate   collects time dependent data
     *
     * count        is used to count incidences only
     *
     * histogram    collects data in histogram form and
     *              prints the end result as a picture
     *
     * tally        collects time independent data
     *
     * attributes shared by these definitions :
     *              reset       note time and reset status
     *                          to zero
     *              update(v)   record new entry v
     *              report      print current status
     *
     * these classes are prefixed by 'tab' which contains
     * common variables and the parameter 'title' which names
     * the particular object.
     * every object of a class inner to tab is put into a
     * 'reportq' behind the scenes.
     * these reportqs are system defined and are called
     *
     *    accumq  countq  dist(+emp)q  histoq  tallyq
     *
     * on a call 'report', the current statuses of all these
     * reportqs are written out.
     *
     * the set of predefined data collection facilities
     * is prefixed by tab.
     *
     * tab          defines the common core
```

```
      *
      * variables:
      *             title       user supplied descriptive text
      *             obs         no. of entries since resetat
      *             resetat     time when initiated, or last reset
      *             next        ref to next tab in reportq
      *
      * procedures:
      *
      *            join        enters this tab into a named
      *                        reportq at the end
      *
      *            reset       (virtual) notes the time in
      *                        resetat and sets obs to zero
      *
      *            writetrn    prints on one (part)line
      *                        title/reset time/observations
      *;


      comment----------------- t a b ---------------------------;

      class tab(title); value title; text title;
              virtual : procedure reset, report;
 b2   begin integer obs;  real resetat;
        ref(tab)next;

        procedure join(r); ref(reportq)r;
 b3     begin
          if r == none then error(18, none, this tab, 0,
             "t.join(r); ref(tab)t; ref(reportq)r;") else
          if r.first == none
            then r.first :- r.last      :- this tab
            else r.last  :- r.last.next :- this tab;
 e3     end***join***;

        procedure report;
 b4     begin
          writetrn;
          outf.outimage;
 e4     end***report***;

        procedure reset;
 b5     begin
          obs     := 0;
          resetat := time;
 e5     end***reset***;

        procedure writetrn;
 b6     begin
          outf.outtext(title);
```

```
          outf.setpos(outf.pos+(13-title.length));
          printreal(resetat);
          outf.outint(obs, 7);
e6      end***report title, resetat and readings***;
```

```
          if title.length > 12 then title :- title.sub(1, 12);
          reset;
 e2     end***tab***;


        comment-------------------- t a l l y -----------------------;

        tab class tally;
 b7     begin                    comment
          *
          * variables:
          *    .title    user supplied descriptive text (parameter)
          *    .obs      number of incidences
          *    .resetat  last time of setting or time of creation
          *    .next     ref to .next tab in reportq
          *     sum      sum of sample values
          *     sumsq    sum of squares of sample values
          *    (variance)(obs*sumsq - sum*sum)/(obs*(obs-1))
          *    (sigma)   sqrt(variance)
          *     min      least sample value
          *     max      largest sample value
          *
          * procedures :
          *    reset     resets obs, sum, sumsq, min, max to zero
          *              copies time into resetat
          *
          *    update(v) adds  1  to  obs
          *              adds  v  to  sum
          *              adds v*v to  sumsq
          *              max becomes maximum (max,v)
          *              min becomes minimum (min,v)
          *
          *    report    prints on one line:
          *              title/reset/obs/av/est.st.dev/min/max
          *;

          real sum, sumsq, min, max;

          procedure report;
 b8       begin writetrn;
            if obs = 0 then outf.outtext(minuses.sub(1, 40)) else
 b9         begin printreal(sum/obs);
              if obs = 1 then outf.outtext(minuses.sub(1, 10)) else
                printreal(sqrt(abs(obs*sumsq-sum**2)/(obs*(obs-1))));
              printreal(min);
              printreal(max);
 e9         end;
            outf.outimage;
 e8       end***report***;

          procedure reset;
```

```
b10    begin obs := 0;
         sum     := sumsq  := min   := max    := 0.0;
         resetat := time;
e10    end***reset***;
```

```
        procedure update(v); real v;
 b11    begin obs := obs + 1;
          sum    := sum + v;
          sumsq  := sumsq + v**2;
          if obs = 1 then min := max := v else
             if v < min then min := v else
             if v > max then max := v;
 e11    end***update***;

        if not(this tally is notally) then join(tallyq);
 e7   end***tally***;


      comment notally is used in histogram. notally objects are
              not entered into tallyq;

      tally class notally;;


      comment------------------ c o u n t -------------------------;

      tab class count;
 b12  begin                   comment
        *
        * variables :
        *    .title        user supplied descriptive text (parameter)
        *    .obs          number of incidences
        *    .resetat      last time of setting or time of creation
        *    .next         ref to next tab in reportq
        *
        * procedures :
        *     .reset       resets obs to zero
        *                  copies time into resetat

        *     update(v)    adds v to obs
        *
        *     report       prints on one line:
        *                  title/reset/observations
        *;

        procedure report;
 b13    begin
          outf.setpos(21);
          writetrn;
          outf.outimage;
 e13    end***report***;

        procedure update(v); integer v;
 b14    begin
          obs  := obs + v;
 e14    end***update***;
```

```
        join(countq);
e12  end***count***;
```

```
      comment-------------------- a c c u m u l a t e -------------;

      tab class accumulate;
 b15  begin                  comment
        *
        * variables :       **** time weighted ****
        *    .title         user supplied descriptive text (param.)
        *    .obs           number of incidences
        *    .resetat       last time of setting or time of creation
        *    .next          ref to next tab in reportq
        *     sumt          time weighted sum
        *     sumsqt        time weighted sum of squares
        *    (mean)         sum/timespan = (last update time-resetat)
        *    (sigma)        sqrt( sumsqt / timespan - mean**2)
        *     min           least sample value
        *     max           largest sample value
        *     lasttime      time of last update
        *     lastv         last update value
        *
        * procedures :
        *     reset         resets obs, sum, sumsqt, min, max to zero
        *                   copies time into resetat, lasttime
        *
        *     update(v)     adds 1 to obs
        *                   adds v*span to sumt
        *                   adds v*v*span to sumsqt
        *                   min becomes minimum(min, v)
        *                   max becomes maximum(max, v)
        *                   copies time into lasttime
        *
        *     report        prints on one line:
        *                   title/reset/obs/mean/est.st.dev./min/max
        *;

        real sumt, sumsqt, min, max, lasttime, lastv;

        procedure report;
 b16    begin real span, avg, t;
          writetrn;
          if obs = 0 then outf.outtext(minuses.sub(1, 40)) else
 b17      begin t := time;
            span := t - resetat;  t := t - lasttime;
            if span<epsilon then outf.outtext(minuses.sub(1,20)) else
 b18        begin avg := (sumt+lastv*t)/span;
              printreal(avg);
              printreal(sqrt(abs((sumsqt+lastv**2*t)/span-avg**2)));
 e18        end;
            printreal(min);
            printreal(max);
 e17      end;
          outf.outimage;
```

```
e16    end***report***;

       procedure reset;
b19    begin obs := 0;
```

```
          sumt := sumsqt := 0.0;
          min  := max    := 0.0;
          lasttime := resetat := time;
 e19    end***reset***;

        procedure update(v); real v;
 b20    begin real now, span;
          obs      := obs + 1;
          now      := time;
          span     := now - lasttime;
          lasttime := now;
          sumt     := sumt + lastv*span;
          sumsqt   := sumsqt + lastv**2*span;
          lastv    := v;
          if obs = 1 then min := max := v else
             if v < min then min := v else
             if v > max then max := v;
 e20    end*** update ***;

        join(accumq);
 e15  end***accumulate***;


      comment-------------------- h i s t o g r a m --------------;

      tab class histogram(lower, upper, ncells); real lower, upper;
                                                 integer ncells;
 b21  begin                  comment
        *
        * variables:
        *
        *    .title        user supplied descriptive text (param.)
        *    .n            number of incidences
        *    .resetat      last time of setting or time of creation
        *    .next         ref to next tab in reportq
        *     lower        lower limit of the variable range
        *     upper        upper limit of the variable range
        *     ncells       number of cells in this range
        *     width        cell width (= (upper - lower)/ncells)
        *     table        array to hold the incidences. values in
        *                  range go in cells 1, 2, ...., n.
        *                  underflow values go in cell 0.
        *                  overflow values in cell limit=ncells+1
        *     limit        ncells + 1.
        *     myt          to accumulate sum , sumsq of readings
        *
        * procedures:
        *     reset        sets obs to zero
        *                  copies time into resetat
        *                  resets myt
        *
```

```
*       update(v)     adds 1 to obs
*                     adds 1 to the appropriate table cell
*                     calls myt.update(v)
*
```

```
        *     report       draws a picture of the histogram.
        *                  calls myt.report
        *;

        integer array table(0 : ncells + 1);
        ref(notally)myt;
        integer limit;
        real width;

        procedure report;
 b22    begin text t;
          integer i, next, a, occ;
          real r, f, scale, sum, freq;

          integer procedure maximumelement;
 b23      begin integer k, j;
            if obs > 0 then
 b24        begin k := table(0);
              for  j := 1 step 1 until limit do
                if table(j) > k then k := table(j);
              maximumelement := k;
 e24        end;
 e23      end*** maximum element ***;

          a   := 40;
          outf.setpos(29);
          outf.outtext("s u m m a r y");
          outf.outimage;  outf.outimage;
          outf.outtext(headingrtn);
          outf.outtext(tallyheading);
          outf.outimage;
          myt.report;
          outf.outimage;
          if obs = 0 then
 b25      begin
            outf.setpos(21);
            outf.outtext("***no entries recorded***");
 e25      end else
 b26      begin scale  := 30 / maximumelement;
            outf.outtext("cell/lower lim/    n/   freq/  cum %");
            outf.outimage;
            outf.setpos(a);  outf.outchar('i');
            outf.outtext(minuses.sub(1, 30));  outf.outimage;
            f     := 1/obs;
            r     := lower - width;
            for i := 0 step 1 until limit do
 b27        begin outf.outint(i, 4);
              if i = 0 then outf.outtext(" -infinity")
                       else printreal(r);
              next := table(i);          outf.outint(next, 6);
              freq := next*f;            outf.outfix(freq, 2, 8);
```

```
                sum  := sum + freq*100.0;  outf.outfix(sum , 2, 8);
                outf.setpos(a);  outf.outchar('i');
                if next > 0 then
b28             begin t :- stars.sub(1, scale*next);
```

```
                if t == notext then outf.outchar('.')
                               else outf.outtext(t);
 e28          end;
              outf.outimage;
            anymoretoprint:
              occ  := occ+next;
              if occ = obs and i+3 < limit then
 b29          begin
                outf.outimage;
                outf.setpos(a+6);
                outf.outtext("**rest of table empty**");
                outf.outimage;
                outf.outimage;
                goto finish;
 e29          end;
              r := r + width;
 e27        end;
          finish:
            outf.setpos(a);  outf.outchar('i');
            outf.outtext(minuses.sub(1, 30));  outf.outimage;
 e26      end;
          outf.outimage;  outf.outimage;
 e22    end***report***;

        procedure reset;
 b30    begin integer k;
          obs := 0;
          for k := 0 step 1 until limit do
            table(k) := 0;
          resetat := time;
          if myt =/= none then myt.reset;
 e30    end***reset***;

        procedure update(v); real v;
 b31    begin integer cell;
          obs   := obs + 1;
          myt.update(v);
          v     := v - lower;
          if v < 0.0 then cell := 0 else
 b32      begin cell  := entier(v/width) + 1;
            if cell > limit then cell := limit;
 e32      end;
          table(cell) := table(cell) + 1;
 e31    end*** update ***;

        if upper <= lower or ncells < 1 then
 b33    begin
          error(19, none, this tab, 0, "new histogram(t,l,u,n);");
          if ncells < 1 then ncells := 10;
          if lower >= upper then
 b34      begin
```

```
          lower := 0.0;
          upper := 100.0;
e34     end;
e33   end;
```

```
        width := (upper - lower)/ncells ;
        limit := ncells + 1;
        myt   :- new notally(title);
        join(histoq);
 e21  end***histogram***;


      comment----------r e g r e s s i o n s-------------;

      tab class regression(title2); value title2; text title2;
 b35  begin
        real x, y, xx, xy, yy;

        procedure update(vx, vy); real vx, vy;
 b36    begin
          obs := obs + 1;
          x   := x + vx;
          y   := y + vy;
          xx  := vx**2 + xx;
          xy  := vx*vy + xy;
          yy  := vy**2 + yy;
 e36    end***update***;

        procedure reset;
 b37    begin
          obs := 0;
          resetat := time;
          x := y := xx := xy := yy := 0.0;
 e37    end***resetat***;

        procedure report;
 b38    begin
          real dx, dy, a0, a1, sd, r2;

          outf.setpos((52-title.length-title2.length)//2);
          outf.outtext("regression of '");
          outf.outtext(title2);
          outf.outtext("' upon '");
          outf.outtext(title);
          outf.outchar(''');
          outf.outimage;  outf.outimage;
          outf.setpos(17);
          outf.outtext("   (re)set/    obs/     xbar/     ybar");
          outf.outimage;
          outf.setpos(17);
          printreal(resetat);
          outf.outint(obs, 8);
          if obs > 0 then
 b39      begin
            printreal(x/obs);
            printreal(y/obs);
```

```
e39     end;
        outf.outimage;  outf.outimage;
        if obs <= 5 then
b40     begin
```

```
           outf.setpos(24);
           outf.outtext("*** insufficient data ***");
 e40      end else
 b41      begin
           dx := abs(obs*xx - x**2);
           dy := abs(obs*yy - y**2);
           if dx < 0.00001 or dy < 0.00001 then
 b42       begin
             outf.setpos(27);
             outf.outtext("***degenerate data***");
             outf.outimage;
             if dx < 0.00001 then
 b43         begin
               outf.setpos(25);
               outf.outtext("x = constant = ");
               printreal(x/obs);
               outf.outimage;
 e43         end;
             if dy < 0.00001 then
 b44         begin
               outf.setpos(25);
               outf.outtext("y = constant = ");
               printreal(y/obs);
               outf.outimage;
 e44         end;
 e42       end***degenerate case***else
 b45       begin
             a1 := (obs*xy - x*y)/dx;
             a0 := (y*xx - x*xy)/dx;
             sd := sqrt((yy - a0*y - a1*xy)/(obs-2));
             r2 := (obs*xy - x*y)**2/(dx*dy);
             outf.outtext("  res.st.dev/ est.reg.coeff/ intercept/");
             outf.outtext(" st.dev.reg.coeff/ corr.coeff");
             outf.outimage;
             outf.setpos( 3);  printreal(sd);
             outf.setpos(18);  printreal(a1);
             outf.setpos(29);  printreal(a0);
             outf.setpos(47);  printreal(obs*sd/sqrt((obs-2)*dx));
             outf.setpos(59);  printreal(sqrt(r2));
             outf.outimage;
 e45       end;
 e41      end;
          outf.outimage;
          outf.outimage;
 e38    end***report***;

        if title2.length > 12 then title2 :- title2.sub(1, 12);
 e35  end***regression***;


      comment--------------------seed generator--------------------
```

```
*
* the basic rng is
*
*    u(k+1) <- u(k) * 2**13 modulo 67099547
```

```
        *
        * (see ]next page: zyqsample in dist)
        * this rng was developed and tested by downham and roberts
        * by noting that
        *
        *    u(k+120633) <- u(k) * 36855 modulo 67099547
        *
        * we get our routine for generating well separated seeds
        *
        * u(0) <- 907, u(1) <- 33427485, u(2) <- 22276755, ...
        *
        * you may change the defaults by assigning a fresh value
        * to zyqseed.
        *;

        integer procedure zyqnextseed;
   b46  begin integer k;
          for k := 7, 13, 15, 27 do
   b47    begin zyqseed := zyqseed*k;
            if zyqseed >= zyqmodulo then
              zyqseed := zyqseed - zyqseed//zyqmodulo*zyqmodulo;
   e47    end;
          zyqnextseed := zyqseed;
   e46  end***zyqnextseed***;

        procedure setseed(n); integer n;
   b48  begin
          if n < 0 then n := -n;
          if n >= zyqmodulo then n := n-n//zyqmodulo*zyqmodulo;
          if n = 0 then n := zyqmodulo//2;
          zyqseed := n;
   e48  end***setseed***;


        comment-------------d i s t r i b u t i o n s----------------
        *
        * this section has the definitions of the sampling mechanisms
        * defined in demos. these definitions are:
        *
        *                               dist
        *
        *            rdist              idist                 bdist
        *
        * rdist =
        * constant   erlang   empirical   negexp    normal    uniform
        *
        * idist =
        * randint   poisson
        *
        * bdist =
        * draw
```

```
* constant          every sample returns the same value.
*
* empirical         defines a cumulative probability function
*                   supplied as a pair of tables by the user.
```

```
      *
      * and the rest follow simula's drawing procedures in the
      * obvious way. by building an object, we make a drawing by a
      * call 'obj'.sample and need not pass over any parameters.
      * and the object name can be relevant, e.g. arrivals.sample.
      *;

      tab class dist;
 b49  begin integer u, ustart, type;
        boolean antithetic;

        real procedure zyqsample;
 b50    begin integer k;
          for k := 32, 32, 8 do
 b51      begin u := k*u;
            if u >= zyqmodulo then u := u - u//zyqmodulo*zyqmodulo;
 e51      end;
          zyqsample := if antithetic then 1.0 -  u/zyqmodulo
                                     else u/zyqmodulo;
          obs := obs+1;
 e50    end***zyqsample***;

        procedure setseed(n); integer n;
 b52    begin
          if n < 0 then n := -n;
          if n >= zyqmodulo then n := n-n//zyqmodulo*zyqmodulo;
          if n = 0 then n := zyqmodulo//2;
          u := n;
 e52    end***setseed***;

        procedure zyqfail(t1,t2,x,y);value t1,t2;text t1,t2;real x,y;
 b53    begin
            switch case:=normall, uniforml, erlangl,
                         randintl, negexpl, poissonl;
          outtext("**error in creation of ");
          outtext(disttype(type));
          outtext("dist '");
          outtext(title);
          outchar(''');  outchar('.');
          outimage;
          outtext(zyqreason); outtext(t1); outimage;
          outtext(zyqrecvry); outtext(t2);
          goto case(type);
          goto join;
            normall:
            erlangl:
            negexpl:
          poissonl:  outreal(x, 5, 12);
                     goto join;
          uniforml:  outreal(x, 5, 12);
                     outtext(", b =");
```

```
            outreal(y, 5, 12);
            goto join;
randintl:   outint(this dist qua randint.a, 10);
            outtext(", b =");
```

```
                    outint(this dist qua randint.b, 10);
         join:      outchar('.'); outimage; outimage;
 e53    end***zyqfail***;

        procedure report;
 b54    begin switch case := normall, uniforml, erlangl, randintl,
                             negexpl, poissonl, drawl,  constantl;
          writetrn;
          outf.outchar(' ');
          outf.outtext(disttype(type));
          outf.setpos(41);
          goto case(type);
          goto skipall;
          normall:    printreal(this dist qua normal.a);
                      printreal(this dist qua normal.b);
                      goto exit;
          uniforml:   printreal(this dist qua uniform.a);
                      printreal(this dist qua uniform.b);
                      goto exit;
          erlangl:    printreal(this dist qua erlang.a);
                      outf.outint(this dist qua erlang.b, 10);
                      goto exit;
          randintl:   outf.outint(this dist qua randint.a, 10);
                      outf.outint(this dist qua randint.b, 10);
                      goto exit;
          negexpl:    printreal(this dist qua negexp.a);
                      goto skip;
          poissonl:   printreal(this dist qua poisson.a);
                      goto skip;
          drawl:      printreal(this dist qua draw.a);
                      goto skip;
          constantl:  printreal(this dist qua constant.a);
                      goto skipall;
          skip:       outf.setpos(61);
          exit:       outf.outint(ustart, 10);
          skipall:    outf.outimage;
 e54    end***report***;

        u := ustart := zyqnextseed;
        if this dist in empirical then join(empq)
                                  else join(distq);
 e49  end***dist***;


      comment--------------------r   d i s t s--------------------;


      dist class rdist; virtual: real procedure sample;;


      rdist class constant(a); real a;
```

```
b55  begin
       real procedure sample;
b56    begin obs := obs + 1;
         sample := a;
```

```
 e56    end***sample***;

        type := 8;
 e55  end***constant***;


      rdist class normal(a, b); real a, b;
 b57  begin real zyqu, zyqv; boolean zyqeven;

        real procedure sample;
 b58    begin real z;
          if zyqeven then
 b59      begin zyqeven := false;
            z    := zyqu*cos(zyqv);
            obs  := obs + 1;
 e59      end else
 b60      begin zyqeven := true;
            zyqu := sqrt(-2.0*ln(zyqsample));
            zyqv := 6.28318530717959*zyqsample;
            z    := zyqu*sin(zyqv);
            obs  := obs - 1;
 e60      end;
          sample := z*b+a;
 e58    end***sample***;

        type := 1;
        if b < 0.0 then
 b61    begin b := -b;
          zyqfail("st. dev. 'b' < 0.0.",
                  "absolute value B! taken. b is now", b, 0.0);
 e61    end;
 e57  end***normal***;


      rdist class negexp(a); real a;
 b62  begin
        real procedure sample;
 b63    begin
          sample := -ln(zyqsample)/a;
 e63    end***sample***;

        type := 5;
        if a <= 0.0 then
 b64    begin a := if a < 0.0 then -a else 0.001;
          zyqfail("non-positive value for 'a' (=arrival rate).",
                  "a reset to", a, 0.0);
 e64    end;
 e62  end***negexp***;


      rdist class uniform(a, b); real a, b;
```

```
b65  begin real zyqspan;
       real procedure sample;
b66    begin
         sample := zyqspan*zyqsample + a;
```

```
 e66    end***sample***;

        type := 2;
        if a > b then
 b67    begin real q;
          q := a; a := b; b := q;
          zyqfail("lower bound 'a' > upper bound 'b'.",
                  "bounds swapped. now, a =", a, b);
 e67    end;
        zyqspan := b-a;
 e65  end***uniform***;


      rdist class erlang(a, b); real a;  integer b;
 b68  begin real zyqab;
        real procedure sample;
 b69    begin integer k, m; real prod;
          m := obs;
          prod := zyqsample;
          for k := 2 step 1 until b do
            prod := prod * zyqsample;
          obs := m+1;
          sample := -ln(prod)*zyqab;
 e69    end***sample***;

        type := 3;
        if a <= 0.0 then
 b70    begin a := if a < 0.0 then -a else 0.01;
          zyqfail("'a' (=1/mean) <= 0.0.",
                  "a reset to", a, 0.0);
 e70    end;
        if b < 0.0 then
 b71    begin b := if b < 0 then -b else 1;
          zyqfail("'b' (erlang st. dev.) <= 0.0.",
                  "b reset to", b, 0.0);
 e71    end;
        zyqab := a/b;
 e68  end***erlang***;


      rdist class empirical(size); integer size;
 b72  begin real array x, p(1 : size);

        real procedure sample;
 b73    begin real q;  integer k;
          q := zyqsample;
          k := 2;
          while p(k) < q do
            k := k + 1;
          sample := x(k-1) + (x(k)-x(k-1))*(q-p(k-1))/(p(k)-p(k-1));
 e73    end***sample***;
```

```
       procedure report;
b74    begin integer k;
         outf.setpos(16);
```

```
          outf.outtext(headingrtn); outf.outtext("/     seed");
          outf.outimage;
          outf.setpos(16);
          writetrn;
          outf.outint(ustart, 10);
          outf.outimage;  outf.outimage;
          outf.setpos(16);
          outf.outtext("       k/  dist. x(k)/  prob. p(k)");
          outf.outimage;
          for k := 1 step 1 until size do
 b75      begin
            outf.setpos(16);
            outf.outint(k, 8);
            outf.outfix(x(k), 5, 13);
            outf.outfix(p(k), 5, 13);
            outf.outimage;
 e75      end;
          outf.outimage;
 e74    end***report***;

        procedure read;
 b76    begin boolean good, first; integer k, l;
          real a, b;

          procedure z(w, r, f, c); value w, c; text w, c;
                                             real r; boolean f;
 b77      begin
            if good then
 b78        begin good := false;
              sysout.setpos(11);
              outtext("**read fault(s) in empirical '");
              outtext(title);
              outchar(''');  outchar('.');
              outimage;
 e78        end;
            if first then
 b79        begin first := false;
              outimage;
              outtext("**inputs   : k ="); outint(k, 4);
              outtext(", dist(k) =");      outfix(a, 3, 10);
              outtext(", prob(k) =");      outfix(b, 3, 10);
              outimage;
              outtext(zyqrecvry);
 e79        end;
            sysout.setpos(14);
            outtext(w);
            if f then outfix(r, 6, 10) else outfix(r, 3, 10);
            outtext(c); outchar('.');
            outimage;
 e77      end***z - the warning routine***;
```

```
k := 1;
good := first := true;
x(1) := a := inf.inreal; b := inf.inreal;
if abs(b) > epsilon then
```

```
             z("p(1) is not zero. p(1) =>",0.0,true," (first prob)");
           for k := 2 step 1 until size do
 b80       begin first := true;
             x(k) := a := inf.inreal; p(k) := b := inf.inreal;
             if a < x(k-1) then
 b81         begin x(k) := x(k-1);
               z("x(k) < x(k-1). x(k) =>",x(k),false," (=x(k-1))");
 e81         end;
             if b < 0.0 or b < p(k-1) or b > 1.0 then
 b82         begin p(k) := p(k-1)+0.001;
               if p(k) > 1.0 then p(k) := 1.0;
               z("illegal prob. p(k) =>",p(k),true," (=p(k-1)+)");
 e82         end;
 e80       end;
           if abs(p(size)-1.0) > epsilon then
            z("p(size) ne 1.0. p(size) =>", 1.0, true, " (last prob.)");
           p(size) := 1.0;
           if not good then
 b83       begin outtext(minuses.sub(1, 62));
             outimage; outimage;
 e83       end;
 e76     end***read***;

         type := 9;
         if size = 1 then error(20, none, this empirical, 0,
             "new empirical(t, size); text t; integer size;");
         read;
 e72   end***empirical***;


       comment--------------------i   d i s t s--------------------;

       dist class idist; virtual: integer procedure sample;;


       idist class randint(a, b); integer a, b;
 b84   begin real zyqspan;
         integer procedure sample;
 b85     begin
           sample := entier(zyqspan*zyqsample) + a;
 e85     end***sample***;

         type := 4;
         if a > b then
 b86     begin integer q;
           q := a; a := b; b := q;
           zyqfail("lower bound 'a' > upper bound 'b'.",
                   "bounds swapped. now a =", a, b);
 e86     end;
         zyqspan := b-a+1;
 e84   end***randint***;
```

```
     idist class poisson(a); real a;
b87  begin
```

```
        integer procedure sample;
 b88    begin integer m; real p, q;
          p := exp(-a);
          q := 1.0;
          l: q := q*zyqsample;
          if q >= p then
 b89      begin
            m := m + 1;
            goto l;
 e89      end;
          sample := m;
          obs := obs - m;
 e88    end***sample***;

        type := 6;
        if a <= 0.0 then
 b90    begin a := if a < 0.0 then -a else 0.001;
          zyqfail("non-positive value for 'a' (=mean).",
                  "a reset to", a, 0.0);
 e90    end;
 e87  end***poisson***;


      comment--------------------b   d i s t s--------------------;

      dist class bdist; virtual: boolean procedure sample;;


      bdist class draw(a); real a;
 b91  begin
        boolean procedure sample;
 b92    begin
          sample := a > zyqsample;
 e92    end***sample***;

        type := 7;
 e91  end***draw***;


      comment-------------readdist-----------------------------------;

      procedure readdist(d, title); name d; value title;
                                 ref(dist)d; text title;
 b93  begin text f, rest;
        integer p, imlength1, l, k, t;

        procedure fail(d, eof); boolean d, eof;
 b94    begin outtext("**error in reading dist with title = '");
          outtext(title);
          outchar(''');  outchar('.');
          outimage;
```

```
outtext("**no match found when scanning input file for ");
if d then outtext(" dist type") else outtext("title");
outchar('.');
outimage;
```

```
          outtext(zyqreason);
          if eof then outtext("end of input file marker hit.") else
 b95      begin outtext("rest of current input image reads:");
            outimage;
            outtext(rest);
 e95      end;
          abort(true);
 e94    end***fail***;

        comment***checktitle***;
        imlength1 := inf.image.length + 1;
        if inf.lastitem then fail(false, true);
        l := title.length;
        p := inf.image.pos;
        rest :- inf.image.sub(p, imlength1 - p);
        if rest.length >= l then f :- rest.sub(1, l);
        if f ne title then fail(false,false);
        inf.setpos(p + l);

        comment***get dist type***;
        if inf.lastitem then fail(true, true);
        p    := inf.image.pos;
        rest :- inf.image.sub(p, imlength1 - p);
        l    := rest.length;
        for k := 6, 7, 6, 7, 6, 7, 4, 8, 9 do
 b96    begin t := t + 1;
          if k <= l then
 b97      begin
            if disttype(t) = rest.sub(1, k) then goto found;
 e97      end;
 e96    end;
        fail(true, false);
        found: inf.setpos(p + k);
        if t=1 then d:-new normal(title,inf.inreal,inf.inreal) else
        if t=2 then d:-new uniform(title,inf.inreal,inf.inreal)else
        if t=3 then d:-new erlang(title,inf.inreal,inf.inint)  else
        if t=4 then d:-new randint(title,inf.inint,inf.inint)  else
        if t=5 then d:-new negexp(title,inf.inreal)            else
        if t=6 then d:-new poisson(title,inf.inreal)           else
        if t=7 then d:-new draw(title,inf.inreal)              else
        if t=8 then d:-new constant(title,inf.inreal)          else
        if t=9 then d:-new empirical(title, inf.inint);
 e93  end***readdist***;


      comment-------------------- reportq ------------------------;

      class reportq(h, l1, l2); value h; text h, l1, l2;
 b98  begin                comment
        *
        * every created tab is put into a reportq in the order
```

```
* of its creations. from there they can all be reported
* together on a call 'report' , or all reset to the null
* state by a call 'reset'.
*
```

```
        * variables :
        *     first    ref to first tab in reportq
        *     last     ref to last tab in reportq
        *
        * procedures:
        *     reset    resets each and every represented tab
        *
        *     report   reports each and every tab as above
        *;

        ref(tab)first, last;

        procedure report;
 b99    begin ref(tab)t;
          integer p, l;
          l := h.length;   p := (72-l)//2;
          outf.setpos(p);  outf.outtext(h);
          outf.outimage;
          outf.setpos(p);  outf.outtext(stars.sub(1, l));
          outf.outimage;   outf.outimage;
          if l1 =/= notext then
 b100     begin
            outf.outtext(l1);
            if l2 =/= notext then outf.outtext(l2);
            outf.outimage;
 e100     end;
          t :- first;
          while t =/= none do
 b101     begin t.report;
            t :- t.next;
 e101     end;
 e99    end***report***;

        procedure reset;
 b102   begin ref(tab)t;
          t :- first;
          while t =/= none do
 b103     begin t.reset;
            t :- t.next;
 e103     end;
 e102   end***reset***;

 e98  end***reportq***;


      comment-------------------- reporting aids -----------------;

      procedure clocktime;
 b104 begin outf.setpos(24);
        outf.outtext("clock time = ");
        printreal(time);
```

```
        outf.outimage;
e104 end***clock time***;
```

```
     procedure box(t); value t; text t;
b105 begin
       outf.outtext(stars);  outf.outimage;
       outf.outchar('*');    outf.setpos(70);
       outf.outchar('*');    outf.outimage;
       outf.outchar('*');
       outf.setpos((72 - t.length)//2);
       outf.outtext(t);
       outf.setpos(70);
       outf.outchar('*');
       outf.outimage;
       outf.outchar('*');    outf.setpos(70);
       outf.outchar('*');    outf.outimage;
       outf.outtext(stars);  outf.outimage;
       outf.outimage;
e105 end***box***;


     text procedure edit(t, k); value t; text t; integer k;
b106 begin text s;
       t :- t.strip;
       if t.length > 10 then t :- t.sub(1, 10);
       edit :- s :- blanks(t.length + 2);
       s := t;
       if k < 0 then k := -k;
       if k > 99 then k := k - k//100*100;
       s.sub(s.length-1, 2).putint(k);
e106 end***edit***;


     procedure printreal(x); real x;
b107 begin
       if x > 0.0 then
b108   begin
         if x > 99999.999 or x < 0.1 then outf.outreal(x, 4, 10)
                                     else outf.outfix (x, 3, 10);
e108   end else
       if x = 0.0 then outf.outfix(x, 3, 10) else
b109   begin
         if x < -9999.999 or x > -0.1 then outf.outreal(x, 3, 10)
                                      else outf.outfix (x, 3, 10);
e109   end;
e107 end***printreal***;


     comment----------e n t i t y-------------------------;

     class entity(title); value title; text title;
                               virtual: label loop;
b110 begin real timein, evtime;
       integer priority, cycle, wants, interrupted;
```

```
ref(queue)currentq;
ref(entity)owner;
boolean terminated;
ref(entity)ll, bl, rl;
```

```
        ref(entity)suc, pred;

        boolean procedure avail;
          avail := owner == none;

        boolean procedure idle;
          idle := evtime < 0.0;

        procedure coopt;
 b111   begin
          if owner =/= none then error(1, this entity, none, 0,
                                             "e.coopt; ref(entity)e;");
          owner :- current;
          if zyqtrace > 0 then
            note(1,"coopts",this entity,currentq,0.0,0);
          if currentq =/= none then out;
 e111   end***coopt***;

        procedure interrupt(n); integer n;
 b112   begin
          interrupted := n;
          if zyqtrace > 0 then
            note(24,"interrupts",this entity,none,0.0,n);
          if current =/= this entity then
 b113     begin
            if currentq =/= none then out;
            if evtime >= 0.0 then cancel;
            insertdelay0;
 e113     end;
 e112   end***interrupt***;

        procedure repeat;
 b114   begin cycle := cycle+1;
          goto loop;
 e114   end***repeat***;

        procedure into(q); ref(queue)q;
 b115   begin ref(entity)e;
          if currentq =/= none then
 b116     begin
            error(3, this entity, currentq, 0, "e.into(q);
 ref(queue)q;");
            out;
 e116     end;
          currentq :- q;
          timein   := time;
          inspect currentq do
 b117     begin qint  := qint + (timein-lastqtime)*length;
            lastqtime := timein;
            length    := length+1;
            if length > maxlength then maxlength := length;
```

```
            e :- last;
            if e == none then first :- last :- this entity else
            if e.priority >= priority then
b118        begin
```

```
                pred :- last;
                last :- last.suc :- this entity;
 e118         end else
 b119         begin
                e :- first;
                while e.priority >= priority do
                  e :- e.suc;
                suc :- e;
                pred :- e.pred;
                if pred == none then first    :- this entity
                                else pred.suc :- this entity;
                suc.pred :- this entity;
 e119         end;
 e117       end otherwise error(2, this entity, none, 0,
                                "e.into(q); ref(queue)q;");
 e115     end***into***;

          procedure out;
 b120     begin real t;
            t := time;
            inspect currentq do
 b121       begin qint  := qint + (t-lastqtime)*length;
              length    := length-1;
              lastqtime := t;
              obs       := obs+1;
              t         := lastqtime-timein;
              if t < epsilon then zeros := zeros+1;
              cum := cum+t;
              if suc == none then last :- pred else suc.pred :- pred;
              if pred == none then first :- suc else pred.suc :- suc;
              suc :- pred :- none;
 e121       end;
            currentq :- none;
 e120     end***out***;

          procedure getserialno;
 b122     begin ref(zyqenttitle)z;
            z :- zyqentnames;
            while (if z == none then false else title ne z.t) do
              z :- z.nexttitle;
            if z == none then z :- new zyqenttitle(title);
            title :- blanks(z.l+2);
            title := z.t;
            title.sub(z.l+1, 2).putint(z.n);
            z.n := z.n+1;
            if z.n = 100 then z.n := 0;
 e122     end***get serialno***;

          ref(entity)procedure nextev;
 b123     begin
            ref(entity)p;
```

```
        p :- bl;
        if p =/= none then
b124    begin
          p :- p.rl;
```

```
              if p =/= none then
 b125         begin
                while p.ll =/= none do
                  p :- p.ll;
 e125         end;
              nextev :- p;
 e124       end;
 e123     end***nextev***;

          procedure list;
 b126     begin
            ref(entity)r;
            integer n;
            outf.setpos(6);
            printreal(evtime);
            outf.outchar(' ');
            outf.outtext(title);
            n := 30;
            for r :- ll, bl, rl do
 b127       begin
              outf.setpos(n);
              if r =/= none then outf.outtext(r.title);
              n := n + 13;
 e127       end;
            outf.outimage;
 e126     end***list***;

          procedure insert;
 b128     begin
                              comment
            * assume:
            *    ll == bl == rl == none
            *    evtime has been set by hold or schedule
            *
            * this routine is called by schedule and hold
            * and inserts o(log n) from the top.
            *
            *;

            ref(entity)w, x;

          newroot:
            if evtime >= root.evtime then
 b129       begin
              ll   :- root;
              root :- root.bl :- this entity;
 e129       end else

          insertbeloww:
 b130       begin
              w :- root;
```

```
          descend:
            x :- w.ll;
            if x == none then
b131        begin
```

```
           insertasllofw:
             w.ll :- this entity;
             bl   :- w;
             if w == current then
b132         begin
               error(15, this entity, none, 0,
                         "e.insert; ref(entity)e;");
               current :- this entity;
               resume(current);
e132         end;
e131       end else
           if evtime < x.evtime then
         insert1belowx:
b133       begin
             w :- x;
             goto descend;
e133       end else
           if evtime = x.evtime then
         insertbetweenxandw:
b134       begin
             w.ll :- x.bl :- this entity;
             bl   :- w;
             ll   :- x;
e134       end else
         insertinrightsubtreeofw:
b135       begin
             x :- w.rl;
             if x == none then
           insertasnewrlofw:
b136         begin
               w.rl :- this entity;
               bl   :- w;
e136         end else
           insert2belowx:
             if evtime < x.evtime then
b137         begin
               w :- x;
               goto descend;
e137         end else
           insert3betweenwandx:
b138         begin
               w.rl :- x.bl :- this entity;
               bl   :- w;
               ll   :- x;
e138         end;
e135       end;
e130     end;
e128   end***insert***;

       procedure insertaftercurrent;
b139   begin
```

```
                  comment
* assume:
*    ll == bl == rl == none
* this routine is called by acquire, release, etc
```

```
          * and inserts o(1) from the bottom.
          *;

          if evtime < 0.0 then
 b140     begin
            ll :- current;
            evtime := current.evtime;
            if current==root then current.bl:-root:-this entity else
 b141       begin
              bl :- current.bl;
              current.bl :- bl.ll :- this entity;
 e141       end;
 e140     end;
 e139   end***insertaftercurrent***;

        procedure insertdelay0;
 b142   begin
                             comment
          * assume:
          *    ll == bl == rl == none
          * this routine is called by acquire, release, etc
          * and inserts o(1) from the bottom.
          *
          *;

          ref(entity)p;
          if evtime < 0.0 then
 b143     begin
            evtime := current.evtime;
            if evtime >= root.evtime then
 b144       begin
              ll :- root;
              root :- root.bl :- this entity;
 e144       end else
 b145       begin
              p :- current.bl;
              while evtime >= p.evtime do
                p :- p.bl;
              ll :- p.ll;
              bl :- p;
              ll.bl :- bl.ll :- this entity;
 e145       end;
 e143     end;
 e142   end***insertdelay0***;

        procedure schedule(t); real t;
 b146   begin
                             comment
          *
          * a call e.schedule(t) does nothing if e
          * is already in the event list(e.evtime >= 0.0).
```

```
* otherwise, it inserts e into the event list
* 'delay' t. there are three separate cases:
*  t < 0.0, e preempts current (0(1) insert)
*  t = 0.0, e follows  current (o(1) insert)
```

```
          *  t > 0.0, insert e from the top
          *;

          ref(entity)e;

          if terminated then error(14, this entity, none, 0,
                       "e.schedule(t); ref(entity)t; real t;");
          if evtime < 0.0 then
 b147     begin
            if zyqtrace > 0 then
              note(2,"schedules",this entity, none,t,0);
            owner :- none;
            if currentq =/= none then out;
          preemptcurrent:
            if t <= now then
 b148       begin
              evtime     := current.evtime;
              bl         :- current;
              current    :- current.ll :- this entity;
              resume(current);
 e148       end else
          insertbehindcurrent:
            if t <= 0.0 then insertdelay0 else
          insertfromthetop:
 b149       begin
              evtime := current.evtime + t;
              insert;
 e149       end;
 e147     end else error(16, this entity, none, 0,
                     "e.schedule(t); ref(entity)e; real t;");
 e146   end***schedule***;

        procedure cancel;
 b150   begin
          comment
          * a call e.cancel deletes e from the event list.
          * it has no effect if e is passive (e.evtime < 0)
          *;
          ref(entity)p;
          if evtime >= 0.0 then
 b151     begin
            if zyqtrace > 0 and not terminated then
              note(23,"cancels",this entity,none,0.0,0);
            evtime := -1.0;
            if ll == none then
 b152       begin
              if this entity == root then
                error(13,this entity,none,0,"e.cancel; ref(entity)e;");
              if this entity == current then
 b153         begin
                if bl.rl == none then
```

```
              blisnewcurrent:
b154            begin
                  current :- bl;
                  current.ll :- none;
```

```
 e154          end else
             leftmostofsubtreeofblisnewcurrent:
 b155          begin
                 p :- bl.rl;
                 while p.ll =/= none do
                 p :- p.ll;
               swingrsubtreetotheleft:
                 bl.ll :- bl.rl;
                 bl.rl :- none;
                 current :- p;
 e155          end;
               bl:- none;
               resume(current);
 e153        end else
           leafbutnotcurrent:
 b156        begin
               if bl.ll == this entity then bl.ll :- rl;
               bl.rl :- none;
               bl    :- none;
 e156        end;
 e152      end else
           if rl == none then comment but ll =/= none;
 b157      begin
             if this entity == root  then root :- ll else
             if this entity == bl.ll then bl.ll :- ll
                                     else bl.rl :- ll;
             ll.bl :- bl;
             bl :- ll :- none;
 e157      end else
 b158      begin comment neither ll nor rl == none;
             p :- rl;
             p.bl :- bl;
             if this entity == root then root :- p else
 b159        begin
               if bl.ll == this entity then bl.ll :- rl
               else bl.rl :- rl;
               bl :- none;
 e159        end;
             while p.ll =/= none do
             p :- p.ll;
             p.ll :- ll;
             ll.bl :- p;
             rl :- ll :- none;
 e158      end;
 e151    end else error(17, this entity, none, 0,
                        "e.cancel; ref(entity)e;");
 e150  end***cancel***;

       if title.length > 10 then title :- title.sub(1, 10);
       getserialno;
       evtime := -1.0;
```

```
detach;
loop:;
inner;
terminated := true;
```

```
        if zyqtrace>0 then note(3,"***terminates",none,none,0.0,0);
        if evtime >= 0.0 then zyqpassivate;
 e110 end***entity***;


      comment------------h o l d  and  p a s s i v a t e----------;

      real procedure time;
       time := current.evtime;


      procedure zyqpassivate;
 b160 begin
        ref(entity)p;
        if current == root then error(15, current, none, 0,
                                           "passivate;");
        p :- current.bl;
        current.bl :- none;
        current.evtime := -1.0;
      locatenewcurrent:
        if p.rl =/= none then
 b161   begin
          p.ll :- p.rl;
          p.rl :- none;
          while p.ll =/= none do
            p :- p.ll;
 e161   end else p.ll :- none;
        current :- p;
        resume(current);
 e160 end***zyqpassivate***;


      procedure passivate;
 b162 begin
        if zyqtrace>0 then note(22,"passivates",none,none,0.0,0);
        zyqpassivate;
 e162 end***passivate***;


      procedure hold(t); real t;
 b163 begin
                           comment
        *
        * delays current by t (t >= 0.0).
        * if t < 0.0, then t := 0.0
        *;

        ref(entity)p;

        if t < 0.0 then t := 0.0;
        if zyqtrace>0 then note(21,"holds for",none,none,t,0);
```

```
        inspect current do
b164    begin
          evtime := evtime+t;
          if root =/= current then
```

```
        moreworktodo:
 b165     begin
          locatenextev:
            if bl.rl == none then p :- bl else
 b166       begin
              p :- bl.rl;
              while p.ll =/= none do
                p :- p.ll;
 e166       end;
          skipifstillcurrent:
            if evtime >= p.evtime then
 b167       begin
              bl.ll :- bl.rl;
              bl.rl :- none;
              bl :- none;
              current :- p;
              insert;
              resume(current);
 e167       end;
 e165     end;
 e164   end;
 e163 end***hold***;


      comment----------- q u e u e ------------------;

      tab class queue;
 b168 begin
                          comment
        *
        * queue objects may be used by anyone as they
        * stand to 'save' entities, but the prime use
        * of this class is to serve as prefix to
        *
        *  res   bin   waitq   condq
        *
        * variables:
        *    .title        user supplied descriptive text
        *    .obs          no. of completed waits in this q
        *    .resetat      time of creation, or last reset
        *    .next         ref to next tab in reportq
        *     length       current no. of entities waiting
        *     zeros        no. of zero (<epsilon) waits
        *     maxlength    maximum qlength since resetat
        *     qint         to record av. qlength
        *     lastqtime    time of last into/out
        *     cum          to record av. wait
        *     first        ref to first entity inq
        *     last         ref to last entity in q
        *
        * procedures:
```

```
*     list          prints out entities in q,
*                   recording their
*                   order, title, priority, and
*                   time of entry
```

```
        *
        *     report       prints on one line:
        *                  title/reset/obs/maxlength/length/
        *                  av.wait/zero waits
        *                  the av.wait includes zeros
        *                  zero wait means in q for < 0.0005
        *
        *     reset        sets zeros, obs to zero
        *                  qint,cum to zero
        *                  maxlength to length
        *                  lastqtime,resetat to time
        *;

        integer length, zeros, maxlength;
        real qint, cum, lastqtime;
        ref(entity)first, last;

        procedure list;
 b169   begin
          ref(entity)e;
          integer l, k;
          l := title.length;
          outf.setpos((52 - l)//2);
          outf.outtext("entities waiting in ");
          outf.outtext(title);
          outf.outimage;
          outf.setpos((52 - l)//2);
          outf.outtext(stars.sub(1, 20+l));
          outf.outimage; outf.outimage;
          outf.setpos(19);
          if length = 0 then
 b170     begin
            outf.outtext("  **** the queue is empty. ****");
            outf.outimage;
 e170     end else
 b171     begin
            outf.outtext("no object     priority entry in q");
            outf.outimage;  outf.outimage;
            k := 0;
            e :- first;
            while e =/= none do
 b172       begin
              outf.setpos(17);
              k := k+1;
              outf.outint(k, 4);
              outf.setpos(22);
              outf.outtext(e.title);
              outf.setpos(33);
              outf.outint(e.priority, 8);
              outf.setpos(42);
              printreal(e.timein);
```

```
                 outf.outimage;
                 e :- e.suc;
e172           end;
e171         end;
```

```
          outf.outimage;
          outf.outimage;
 e169   end***list***;

        procedure report;
 b173   begin
          real span, t;
          writetrn;
          if (if this queue in condq
                 then this queue qua condq.all
                 else this queue in waitq)
            then outf.image.sub(13,1).putchar('*');
          outf.outint(maxlength, 6);
          outf.outint(length, 6);
          t    := time;
          span := t - resetat;
          outf.setpos(44);
          if span<epsilon then outf.outtext(minuses.sub(1,10)) else
            printreal((qint + (t-lastqtime)*length)/span);
          outf.outint(zeros, 6);
          outf.setpos(61);
          if obs > 0 then printreal(cum/obs)
                     else outf.outtext(minuses.sub(1,10));
          outf.outimage;
          if this queue is waitq then outf.outimage;
 e173   end***report***;

        procedure reset;
 b174   begin
          zeros     := obs  := 0;
          qint      := cum  := 0.0;
          maxlength := length;
          lastqtime := resetat:= time;
 e174   end***reset***;

        if this queue is queue then join(queueq);
 e168 end***queue***;


      queue class noqueue;;


      comment------------r e s o u r c e-------------;

      queue class resource(avail); integer avail;
 b175 begin
                           comment
        * defines the common core to res and bin
        *
        * variables:
        *    .as class queue
```

```
*     avail         amount of resource currently free
*     sint          to maintain usage*time integral
*     extreme       min value of avail if res
*                   max value of avail if bin
```

```
        *     initial      initial value of the resource
        *     lastrtime    time of last acquire/return of res
        *                  or last take/get of bin
        *
        * procedures:
        *     report       prints on one line
        *        res =title/reset/obs/limit/min/now/%usage/av.wait/qmax
        *        bin =title/reset/obs/init/max/now/av.free/wait/qmax
        *
        *     reset        sets obs, zeros, users to zero
        *                  sint, qint, cum to zero
        *                  lastqtime, resetat to time
        *                  maxlength to length
        *                  extreme to avail
        *;
        integer extreme, initial, users;
        real sint, lastrtime;

        procedure report;
 b176   begin real t, span, x;
          t    := time;
          span := t- resetat;
          writetrn;
          fudge: outf.image.sub(24, 7).putint(users);
          outf.outint(initial, 5);
          outf.outint(extreme, 5);
          outf.outint(avail, 5);
          x := sint + (t-lastrtime)*avail;
          if span < epsilon then outf.outtext(minuses.sub(1, 10)) else
          if this resource is bin then printreal(x/span) else
             printreal((1.0-x/(initial*span))*100.0);
          if obs = 0 then outf.outtext(minuses.sub(1,10))
                     else printreal(cum/obs);
          outf.outint(maxlength,5);
          outf.outimage;
 e176   end***report***;

        procedure reset;
 b177   begin
          obs := zeros := users := 0;
          maxlength := length;
          lastrtime := lastqtime := resetat := time;
          qint := sint := cum := 0.0;
          extreme  := avail;
 e177   end***reset***;

        initial := avail;
 e175 end***resource***;


      comment------------r e s----------------------;
```

```
          resource class res;
b178 begin
                                              comment
```

```
        * a res object makes mutual exclision available in demos.
        * an object with limit = n > 0 can be 'used' by up to
        * n entities at a time, but no more. it can be seized
        * in integer chunks (0 < chunk <= limit), and returned
        * all at once or in part chunks. if the amount
        * requested is not free, the requester is delayed.
        * when an entity returns units to the res, the
        * queue of blocked entities is tested from the
        * front.
        *
        * variables:
        *    .title        user supplied descriptive text
        *    .obs          no. of completed usages (calls on return)
        *    .resetat      time of creation, or last reset.
        *    .next         ref to next tab in reportq
        *    .avail        amount currently free
        *    .sint         maintains time weighted average of
        *                   resource usage
        *    .extreme      minimum level reached
        *     initial      maximum level
        *
        * procedures:
        *     acquire(n)   success if n <= avail and
        *                     current.priority > first.priority
        *                  and then avail := avail - n.
        *                  else current is blocked in this queue
        *                 *errors: n < 0
        *                          n > limit
        *
        *     return(n)    avail := avail + n
        *                  activate first delay 0.0
        *                  (first will himself see if he can go)
        *                 *errors: n < 0
        *                          n > limit
        *
        *    .report       see resource
        *
        *    .reset        see resource
        *
        *;

        procedure acquire(m); integer m;
 b179   begin
          real t;
          if m < 1 or m > initial then
            error(if m < 1 then 4 else 5, none, this res, m,
                  "r.aquire(n); ref(res)r; integer n;");
          current.into(this queue);
          current.wants := m;
          if m > avail or current =/= first then
 b180     begin
```

```
if zyqtrace>0 then note(4,"awaits",none,this res,0.0,m);
zyqpassivate;
while m > avail or current =/= first do
  zyqpassivate;
```

```
 e180     end;
          if zyqtrace>0 then note(5,"seizes",none,this res,0.0,m);
          t      := time;
          sint   := sint + (t-lastrtime)*avail;
          lastrtime := t;
          avail := avail - m;
          if avail < extreme then extreme := avail;
          current.out;
          current.wants := 0;
          if (if first == none then false else first.wants <= avail)
            then first.insertaftercurrent;
 e179   end***acquire***;

        procedure release(m); integer m;
 b181   begin
          real t;
          if m < 1 or (m+avail) > initial then
            error(if m < 1 then 7 else 8, none, this res, m,
                  "r.release(n); ref(res)r; integer n;");
          if zyqtrace>0 then note(6,"releases",none,this res,0,m);
          t      := time;
          sint   := sint + (t-lastrtime)*avail;
          lastrtime := t;
          avail := avail + m;
          users := users + 1;
          if (if first == none then false else first.wants <= avail)
            then first.insertdelay0;
 e181   end***release***;

        if avail < 1 then
          error(10, none, this res, avail,
            "new res(title, lim); text title; integer lim;");
        join(resq);
 e178 end***res***;


      comment------------b i n-----------------------;

      resource class bin;
 b182 begin
                          comment
        * class bin caters for for the producer/consumer
        * cooperation : the producer gives, the consumer
        * takes.
        *
        * variables:
        *    .title        user supplied descriptive text
        *    .resetat      time of creation, or last reset
        *    .obs          no. of completed usages(calls on give)
        *    .next         ref to next tab in binq
        *    .avail        amount currently frce
```

```
*    .sint          keeps time-weighted average of bin usage
*    .extreme       maximum level attained
*     initial       initial value of avail
*
```

```
        * procedures:
        *     give(m)      increments avail by m
        *                  activates first
        *                  *errors : m <= 0
        *
        *     take(m)      blocks current if can't proceed
        *                   (first in 0 and avail > = m)
        *                  when can proceed, leaves q
        *                  decrements avail and activates
        *                  first
        *                 *error : m <= 0.
        *
        *    .reset        see resource
        *
        *    .report       see resource
        *;

        procedure take(m); integer m;
 b183   begin
          real t;
          if m < 1 then
            error(6, none, this bin, 0,
                  "b.take(m); ref(bin)b; integer m;");
          current.into(this queue);
          current.wants := m;
          if current =/= first or m > avail then
 b184     begin
            if zyqtrace>0 then note(7,"awaits",none,this bin,0.0,m);
            zyqpassivate;
            while current =/= first or m > avail do
              zyqpassivate;
 e184     end;
          current.out;
          current.wants := 0;
          if zyqtrace>0 then note(8,"seizes",none,this bin,0.0,m);
          t     := time;
          sint  := sint + (t-lastrtime)*avail;
          lastrtime := t;
          avail := avail - m;
          if (if first == none then false else first.wants <= avail)
            then first.insertaftercurrent;
 e183   end***take***;

        procedure give(m); integer m;
 b185   begin
          real t;
          if m < 1 then error(9, none, this bin, m,
                       "b.give(n); ref(bin)b;  integer n;");
          if zyqtrace>0 then note(9,"gives",none,this bin,0.0,m);
          t := time;
          sint := sint + (t-lastrtime)*avail;
```

```
lastrtime := t;
avail := avail + m;
users := users + 1;
if avail > extreme then extreme := avail;
```

```
          if (if first == none then false else first.wantsgive <= avail)
            then first.insertdelay0;
 e185   end***give***;

        if avail < 0 then error(11, none, this bin, avail,
          "new bin(title, init); text title; integer init;");
        join(binq);
 e182 end***bin***;


      comment---------------w a i t q-------------------;

      queue class waitq;
 b186 begin
        ref(queue)masterq;

        procedure wait;
 b187   begin
          current.into(this queue);
          if zyqtrace>0 then note(14,"waits",none,this queue,0.0,0);
          if masterq.first =/= none then masterq.first.insertdelay0;
          zyqpassivate;
 e187   end***wait***;

        ref(entity)procedure coopt;
 b188   begin
          ref(entity)p;
          current.into(masterq);
          if length = 0 or current =/= masterq.first then
 b189     begin
            if zyqtrace>0 then note(17,"waits",none,masterq,0.0,0);
            zyqpassivate;
            while length = 0 do
              zyqpassivate;
            p :- current.suc;
            if p =/= none and length > 1 then p.insertaftercurrent;
 e189     end;
          current.out;
          p :- coopt :- first;
          p.coopt;
 e188   end***coopt***;

        boolean procedure avail(e, c); name e, c;
                          ref(entity)e; boolean c;
 b190   begin
          ref(entity)p;
          e :- p :- first;
          while (if p == none then false else not c) do
            p :- e :- p.suc;
          avail := p =/= none;
 e190   end***avail***;
```

```
       procedure find(e, c); name e, c;
               ref(entity)e; boolean c;
b191   begin
```

```
          ref(entity)p;
          current.into(masterq);
          if not avail(e, c) then
 b192     begin
            if zyqtrace > 0 then
              note(15,"is blocked",none,this waitq,0.0,0);
            zyqpassivate;
            while not avail(e, c) do
 b193       begin
              if current.suc =/= none and length > 0 then
                 current.suc.insertaftercurrent;
              zyqpassivate;
 e193       end;
 e192     end;
          p :- current.suc;
          current.out;
          e.coopt;
          if zyqtrace>0 then note(16,"finds",e,this waitq,0.0,0);
          if p =/= none and length > 0 then p.insertaftercurrent;
 e191   end***find***;

        masterq :- new noqueue(title);
        masterq.join(waitqq);
        join(waitqq);
 e186 end***waitq***;


      comment---------------c o n d q-------------------;

      queue class condq;
 b194 begin
                          comment
        * condq supplies the waituntil capability to demos.
        * an entity waiting until tests the condition at
        * once: if true, it proceeds without delay.
        * otherwise it is delayed until signalled to go on.
        * if all is set, signal checks each and every entity
        * waiting until. else only those at the head of the
        * queue are tested.
        *
        * variables:
        *    .as queue
        *     all         signal tests all waiting entities if set
        *
        * procedures:
        *     waituntil   keeps an entity in this condq
        *                  until the condition is fulfilled
        *
        *     signal      issued by an entity on freeing system
        *                  resources. tests dormant entities
        *;
```

```
boolean all;

procedure waituntil(c); name c; boolean c;
```

```
 b195   begin
          current.into(this condq);
          if not c then
 b196     begin
            if zyqtrace > 0 then
              note(19,"w'until in", none,this condq,0.0,0);
            zyqpassivate;
            while not c do
 b197       begin
              if all and current.suc =/= none then
                current.suc.insertaftercurrent;
              zyqpassivate;
 e197       end;
            if zyqtrace>0 then note(20,"leaves",none,this condq,0.0,0);
 e196     end;
          if current.suc=/=none then current.suc.insertaftercurrent;
          current.out;
 e195   end***waituntil***;

        procedure signal;
 b198   begin
          if zyqtrace > 0 then
            note(25, "signals",  none, this condq, 0.0, 0);
          if length > 0 then first.insertdelay0;
 e198   end***signal***;

        join(condqq);
 e194 end***condq***;


      comment----------t r a c i n g   r o u t i n e s------------;

      real zyqnotelastt;  ref(entity)zyqnotelaste;  integer zyqtrace;

      procedure trace;
      if zyqtrace > 0 then zyqtrace := zyqtrace+1 else
 b199 begin clocktime;
        box("t r a c i n g   c o m m e n c e s");
        outf.outtext("      time/ current     and its action(s)");
        outf.outimage;  outf.outimage;
        zyqtrace := 1;
 e199 end***trace***;

      procedure notrace;
      if zyqtrace > 1 then zyqtrace := zyqtrace-1 else
 b200 begin outf.outimage;  outf.outimage;
        clocktime;
        box("t r a c i n g   s w i t c h e d   o f f");
        zyqtrace := 0;
        zyqnotelastt := -15.0;
        zyqnotelaste :- none;
```

```
e200 end***notrace***;

     procedure note(index,action,e,l,t1,n);value action;text action;
                 integer index, n; real t1; ref(entity)e; ref(tab)l;
```

```
 b201 begin real t; ref(entity)c;

        procedure intout(n); integer n;
 b202   begin integer p;
          if n < 0 then
 b203     begin
            n := -n;
            outf.outchar('-');
 e203     end;
          p := if n < 10 then 1 else
               if n < 100 then 2 else
               if n < 1000 then 3 else
               if n < 10000 then 4 else
               if n < 100000 then 5 else 10;
          outf.outint(n, p);
 e202   end***intout***;

        procedure realout(x); real x;
 b204   begin integer p;
          if x < 0 then
 b205     begin
            x := -x;
            outf.outchar('-');
 e205     end;
          p := if x < 10.0 then 5 else
               if x < 100.0 then 6 else
               if x < 1000.0 then 7 else
               if x < 10000.0 then 8 else
               if x < 100000.0 then 9 else 0;
          if p = 0 then outf.outreal(x, 5, 10) else outf.outfix(x, 3,
 p);
 e204   end***realout***;

        switch message := m1, m2, m3, m4, m5, m6, m7, m8, m9,
                          m10,m11,m12,m13,m14,m15,m16,m17,m18,
                          m19,m20,m21,m22,m23,m24,m25;
        t := time;
        c :- current;
        if (abs(t)-zyqnotelastt) > 0.0005 then
 b206   begin zyqnotelastt := t;
          printreal(t);
 e206   end;
        if zyqnotelaste =/= c then
 b207   begin outf.setpos(12);
          zyqnotelaste :- c;
          outf.outtext(c.title);
 e207   end;

        outf.setpos(25);
        outf.outtext(action);
        outf.outchar(' ');
```

```
goto message(index);

m1:      comment e.coopt;
m18:     comment q.coopt      - coopts;
```

```
m23:    comment cancel(e);
        if e == none then outf.outtext("none!) else
        if e == current then outf.outtext("itself")
                        else outf.outtext(e.title);
        if l =/= none then
        begin outf.outtext(" from ");
          outf.outtext(l.title);
        end;
        goto exit;

m2:     comment e.schedule(t);
        outf.outtext(e.title);
        if t1 <= 0.0 then outf.outtext(" now") else
        begin outf.outtext(" at ");
          realout(t+t1);
        end;
        goto exit;

m3:     comment terminates;
        goto exit;

m4:     comment res.acquire - blocked;
m5:     comment res.acquire - seizes;
m6:     comment res.release - releases;
m7:     comment bin.take    - blocked;
m8:     comment bin.take    - seizes;
m9:     comment bin.give    - releases;
        intout(n);
        if index=6 or index=9 then outf.outtext(" to ")
                              else outf.outtext(" of ");
        outf.outtext(l.title);
        goto exit;

m16:    comment q.find      - finds;
        outf.outtext(e.title);
        outf.outchar(' ');

m14:    comment q.wait;
m15:    comment q.find      - blocked;
m17:    comment q.coopt     - blocked;
        outf.outtext("in ");

m19:    comment q.waituntil - waits;
m20:    comment q.waituntil - leaves;
m25:    comment q.signal;
        outf.outtext(l.title);
        goto exit;

m21:    comment holds;
        realout(t1);
        outf.outtext(", until ");
```

```
        realout(t+t1);
        goto exit;

m22:    comment zyqpassivate;
```

```
                goto exit;

        m24:    comment e.interrupt(n);
                outf.outtext(e.title);
                outf.outtext(", with n = ");
                intout(n);
                goto exit;

          m10:m11:m12:m13:
        exit:   outf.outimage;
 e201 end***note***;

      procedure error(no, e, q, n, call);
               value call; integer no, n;
               text call; ref(entity)e; ref(tab)q;
 b208 begin

        procedure nextline;
 b209   begin
          outchar('.');
          outimage;
          sysout.setpos(9);
 e209   end***nextline***;

        procedure intout(n); integer n;
 b210   begin integer p;
          outchar(' ');
          if n < 0 then
 b211     begin
            n := -n;
            outchar('-');
 e211     end;
          p := if n < 10 then 1 else
               if n < 100 then 2 else
               if n < 1000 then 3 else
               if n < 10000 then 4 else
               if n < 100000 then 5 else 10;
          outint(n, p);
 e210   end***intout***;

        procedure printreal(x); real x;
 b212   begin integer p;
          outchar(' ');
          if x < 0 then
 b213     begin
            x := -x;
            outchar('-');
 e213     end;
          p := if x < 10.0 then 5 else
               if x < 100.0 then 6 else
               if x < 1000.0 then 7 else
```

```
                 if x < 10000.0 then 8 else
                 if x < 100000.0 then 9 else 0;
            if p = 0 then outreal(x, 5, 10) else outfix(x, 3, p);
e212    end***printreal***;
```

```
        ref(entity)c;
        switch case := e1, e2, e3, e4, e5, e6, e7, e8, e9, e10,
                       e11,e12,e13,e14,e15,e16,e17,e18,e19,e20;

        sysout.setpos(23);
        outtext("clock time = ");
        if time > 99999.0 then outreal(time, 5, 12)
                          else outfix(time, 3, 10);
        outimage;
        abort(false);

        outtext("**cause : call on '");
        outtext(call);
        outchar(''');
        nextline;

        outtext("current == ");
        outtext(current.title);
        nextline;
        goto case(no);

        e1:         comment e.coopt;
            outtext("attempt by current to coopt '");
            outtext(e.title);
            outchar(''');
            nextline;
            outtext(e.title);
            outtext(" is already coopted by ");
            outtext(e.owner.title);
            goto blowup;

        e2:     comment e.into(q);
            outtext("q == none");
            nextline;
            outtext("attempt by current to place ");
            if e == current then outtext("itself")
                    else outtext(e.title);
            outtext(" into a null queue");
            nextline;
            outtext("statement ignored.");
            goto continue;

        e3:     comment e.into(q);
            outtext("current tries to place ");
            if e == current then outtext("itself")
                    else outtext(e.title);
            outtext(" into queue '");
            outtext(q.title);  outchar(''');
            nextline;
            outtext(e.title);
            outtext(" is already in ");
```

```
outtext(e.currentq.title);
nextline;
outtext(e.title);
outtext(" leaves ");
```

```
            outtext(e.currentq.title);
            outtext(" and enters ");
            if q == none then outtext("a null queue!)
                     else outtext(q.title);
            goto continue;

        e4:     comment r.acquire(n) : n < 0;
        e5:     comment r.acquire(n) : n > r.limit;
        e6:     comment b.take(n)    : n < 0;
        e7:     comment r.release(n) : n < 0;
        e8:     comment r.release(n) : n > r.limit;
        e9:     comment b.give(n)    : n < 0;
            outtext("n =");
            intout(n);
            nextline;
            if no <= 6 then outtext("request for ")
                   else outtext("attempt to return ");
            if n < 1 then outtext("non-positive")
                 else outtext("too many");
            outtext(" units ");
            if no <= 6 then outtext("from ") else outtext("to ");
            if q in res then outtext("res") else outtext("bin");
            outchar(' ');  outchar(''');
            outtext(q.title);  outchar(''');
            nextline;
            if q in bin then outtext("set n > 0") else
            begin
              outtext("set 0 < n <= r.limit (=");
              intout(q qua resource.initial);
              outchar(')');
            end;
            goto blowup;

        e10:    comment new res(title, limit);
        e11:    comment new bin(title, initial size);
            if no = 10 then outtext("limit of res '")
               else outtext("initial size of bin '");
            outtext(q.title);
            outtext("' =");
            intout(n);
            nextline;
            outtext("it should be ");
            if no=10 then outtext("positive")
                 else outtext("non-negative");
            outchar('.');  outchar(' ');
            if no=10 then outtext("set limit > 0")
                 else outtext("set initial size >= 0");
            goto blowup;

        e12:    comment x.cancel : x idle;
            outtext("entity '");
```

```
outtext(e.title);
outtext("' is not in the event list.");
outimage;
goto continue;
```

```
e13:    comment x.cancel : x sole entity in event list;
    outtext("attempt to cancel last entity in event list");
    goto blowup;

e14:    comment e.schedule(t) : e terminated;
    outtext("e == '");
    outtext(e.title);
    outtext("' is terminated and cannot be scheduled");
    goto blowup;

e15:    comment implementation error;
    outtext("system error: please contact the  ");
    outtext("implementor, graham birtwistle.");
    goto blowup;

e16:    comment e.schedule(t): ref(entity)t: real t;
    outtext("e == '");
    outtext(e.title);
    outtext("' is already scheduled.");
    goto join;

e17:    comment e.cancel: ref(entity)e;
    outtext("attempt to cancel non-scheduled entity e == '");
    outtext(e.title);  outchar(''');
join:   nextline;
    outtext("statement ignored.");
    goto continue;

e18:    comment t.join(r): ref(tab)t: ref(reportq)r;
    outtext("tab '");
    outtext(q.title);
    outtext("' tries to join a null reportq");
    nextline;
    outtext("t will not be reported unless you ");
    outtext(" call 't.report'.");
    goto continue;

e19:    comment new histogram(t, l, u, n);
    outtext("attempt to create illegal histogram '");
    outtext(q.title);
    outchar(''');
    nextline;
    outtext("lower bound = ");
    printreal(q qua histogram.lower);
    nextline;
    outtext("upper bound = ");
    printreal(q qua histogram.upper);
    nextline;
    outtext("upper must be greater than lower");
    nextline;
    outtext("action: lower <- 0.0, and upper <- 100.0.");
```

```
    goto continue;

e20:    comment new empirical(t, 1);
    outtext("attempt to create empirical object '");
```

```
            outtext(q.title);  outchar(''');
            nextline;
            outtext("size = 1. set size > 1");
            goto blowup;

        blowup:     outchar('.'); outimage;
                abort(true);
        continue:   outimage; outimage;
 e208 end***error***;

      procedure abort(b); boolean b;
 b214 begin
        integer l;
        if b then l := 27 else l := 28;
        outtext(minuses);  outimage;
        outtext(minuses.sub(1, l));
        if b then outtext("program  aborted")
             else outtext("serious  error");
        outtext(minuses.sub(1, l));  outimage;
        outtext(minuses);  outimage;
        if b then
 b215   begin
           if inf =/= sysin  and inf.image =/= notext then inf.close;
          if outf =/= sysout and outf.image =/= notext then outf.close;
          l := 0;
          l := 1/l;
 e215   end;
 e214 end***abort***;


      comment---------s n a p p i n g   r o u t i n e s------------;

      procedure report;
 b216 begin ref(reportq)r;
        outf.outimage;  outf.outimage;
        clocktime;
        box("r e p o r t");
        for r:-distq,empq,accumq,countq,tallyq,histoq,regressq,
               resq,binq,queueq,waitqq,condqq do
        if r.first =/= none then
 b217   begin
          outf.outimage;
          outf.outimage;
          r.report;
 e217   end;
 e216 end***report***;

      procedure noreport;
        zyqreport := false;

      procedure reset;
```

```
b218 begin ref(reportq)r;
       for r:-distq,empq,accumq,countq,tallyq,histoq,regressq,
             resq,binq,queueq,waitqq,condqq do
       if r.first =/= none then r.reset;
```

```
 e218 end***reset***;

      procedure snapqueues;
 b219 begin ref(tab)q; integer k; ref(entity)e;
        outf.outimage; outf.outimage;
        clocktime;
        box("l i s t   o f   p a s s i v e   o b j e c t s");
        for q :- queueq.first, condqq.first, waitqq.first do
 b220   begin
          while q =/= none do
 b221     begin
            q qua queue.list;
            outf.outimage;  outf.outimage;
            q :- q.next;
 e221     end;
          outf.outimage; outf.outimage;
 e220   end;
 e219 end***snapqueues***;

      procedure snapsqs;
 b222 begin integer k;
        procedure traverse(r); ref(entity)r;
 b223   begin
          if r.ll =/= none then traverse(r.ll);
          if r.rl =/= none then traverse(r.rl);
          k := k+1;  outf.outint(k, 5);  r.list;
 e223   end***traverse***;

        outf.outimage; outf.outimage;
        clocktime;
        box("e v e n t   l i s t");
        outf.outtext("    n/ ev. time/obj. title / ");
        outf.outtext("ll           bl           rl");
        outf.outimage;
        traverse(root);
        outf.outimage;  outf.outimage;
 e222 end***snapsqs***;

      entity class mainprogram;
 b224 begin
      loop:
        detach;
        goto loop;
 e224 end***mainprogram***;

      class zyqenttitle(t); text t;
 b225 begin integer n, l;
        ref(zyqenttitle)nexttitle;
        nexttitle   :- zyqentnames;
        zyqentnames :- this zyqenttitle;
        l := t.length;
```

```
        n := 1;
e225 end***zyqenttitle***;
```

```
comment--------local variables and their initialisations ----;

ref(reportq)empq, tallyq, accumq, histoq, countq, distq;
ref(reportq)resq, binq, queueq, condqq, waitqq, regressq;
text tallyheading, accumheading, distheading;
text headingrtn, stars, minuses, zyqreason, zyqrecvry;
text resheading, binheading, qheading;
text array disttype(0:9);
integer zyqseed, zyqmodulo;
ref(infile)inf;
ref(outfile)outf;
ref(zyqenttitle)zyqentnames;
ref(mainprogram)demos;
ref(entity)root, current, zyqp;
real now, simperiod, epsilon;
boolean zyqreport;

zyqreport := true;

epsilon := 0.00001;

headingrtn  :-copy("title       /   (re)set/   obs");
accumheading:-copy("/  average/est.st.dv/  minimum/  maximum");
distheading :-copy("/type     /        a/        b/     seed");
tallyheading:-accumheading;
resheading  :-copy("/ lim/ min/ now/  % usage/ av. wait/qmax");
binheading  :-copy("/init/ max/ now/ av. free/ av. wait/qmax");
qheading    :-copy("/ qmax/ qnow/ q average/zeros/  av. wait");

disttype(0) :-copy("undefined");
disttype(1) :-copy("normal");   disttype(2) :-copy("uniform");
disttype(3) :-copy("erlang");   disttype(4) :-copy("randint");
disttype(5) :-copy("negexp");   disttype(6) :-copy("poisson");
disttype(7) :-copy("draw");     disttype(8) :-copy("constant");
disttype(9) :-copy("empirical");

accumq :- new reportq("a c c u m u l a t e s",
                          headingrtn, accumheading);
countq :- new reportq("c o u n t s", blanks(20), headingrtn);
distq  :- new reportq("d i s t r i b u t i o n s",
                          headingrtn, distheading );
empq   :- new reportq("e m p i r i c a l s", notext, notext);
regressq:-new reportq("r e g r e s s i o n s",notext,notext);
histoq :- new reportq("h i s t o g r a m s", notext, notext);
tallyq :- new reportq("t a l l i e s",headingrtn,tallyheading);
resq   :- new reportq("r e s o u r c e s",headingrtn,
                                          resheading);
binq   :- new reportq("b i n s",headingrtn,binheading);
queueq :- new reportq("q u e u e s",headingrtn,qheading);
condqq :- new reportq("c o n d i t i o n   q u e u e s",
                          headingrtn, qheading);
```

```
waitqq :- new reportq("w a i t   q u e u e s",
                          headingrtn, qheading);

stars :- blanks(70);
```

```
      while stars.more do
        stars.putchar('*');

      minuses :- blanks(70);
      while minuses.more do
        minuses.putchar('-');

      inf       :- sysin;
      outf      :- sysout;
      zyqreason :- copy("**reason   : ");
      zyqrecvry :- copy("**recovery : ");
      zyqmodulo := 67099547;  zyqseed := 907;
      now := -10&20;
      zyqnotelastt := -15.0;
      current :- root  :- demos :- new mainprogram("demos");
      demos.evtime := 0.0;
      demos.title :- demos.title.sub(1, 5);
      inner;
      zyqp :- demos.nextev;
      while (if zyqp == none then false else zyqp.evtime = time) do
 b226 begin
        hold(0.0);
        zyqp :- demos.nextev;
 e226 end;
      if zyqreport then report;
       if inf =/= sysin  and inf.image  =/= notext then inf.close;
      if outf =/= sysout and outf.image =/= notext then outf.close;
 e1 end***demos***;
```

appendix b

2 1 d e m o s s a m p l e r u n s

-------------------- program 0 --------------------

```
begin external class demos;
demos
  begin
    noreport;
  end;
end;
```

-------------------- program 1 --------------------

```
begin external class demos;
demos
  begin
    ref(res)tugs, jetties;

    entity class boat;
    begin
    dock:
      jetties.acquire(1);  tugs.acquire(2);
      hold(2.0);
      tugs.release(2);
    unload:
      hold(14.0);
    leave:
      tugs.acquire(1);
      hold(2.0);
      tugs.release(1);  jetties.release(1);
    end***boat***;

    outf :- new outfile("o dske:p1");
    outf.open(blanks(70));
    trace;
    tugs      :- new res("tugs", 3);
    jetties   :- new res("jetties", 2);
    new boat("boat").schedule(0.0);
    new boat("boat").schedule(1.0);
    new boat("boat").schedule(15.0);
    hold(36.0);
  end;
end;
```

-------------------- program 2 --------------------

```
begin external class demos;
demos
  begin
    ref(res)tugs, jetties;
    ref(rdist)next, discharge;

    entity class boat;
    begin
      new boat("boat").schedule(next.sample);
    dock:
      jetties.acquire(1);  tugs.acquire(2);
      hold(2.0);
      tugs.release(2);
    unload:
      hold(discharge.sample);
    leave:
      tugs.acquire(1);
      hold(2.0);
      tugs.release(1);  jetties.release(1);
    end***boat***;

    outf :- new outfile("o dske:p2");
    outf.open(blanks(70));
    trace;
    tugs      :- new res("tugs", 3);
    jetties   :- new res("jetties", 2);
    next      :- new negexp("next boat", 0.1);
    discharge :- new normal("discharge", 14.0, 3.0);
    new boat("boat").schedule(0.0);
    hold(50.0);
    notrace;
    hold(28.0*24.0-50.0);
  end;
end;
```

-------------------- program 3 --------------------

```
begin external class demos;
demos
  begin
    ref(res)tugs, jetties;
    ref(rdist)next, discharge;

    entity class boat;
    begin
      new boat("boat").schedule(next.sample);
    dock:
      tugs.acquire(2);  jetties.acquire(1);
      hold(2.0);
      tugs.release(2);
    unload:
      hold(discharge.sample);
    leave:
      tugs.acquire(1);
      hold(2.0);
      tugs.release(1);  jetties.release(1);
    end***boat***;

    outf :- new outfile("o dske:p3");
    outf.open(blanks(70));
    trace;
    tugs      :- new res("tugs", 3);
    jetties   :- new res("jetties", 2);
    next      :- new negexp("next boat", 0.1);
    discharge :- new normal("discharge", 14.0, 3.0);
    new boat("boat").schedule(0.0);
    hold(50.0);
    notrace;
    hold(28.0*24.0-50.0);
  end;
end;
```

-------------------- program 4 --------------------

```
begin external class demos;
  demos
    begin
      ref(res)file;
      ref(count)reads, writes;

      entity class reader;
      begin
      read:
        file.acquire(1);
        hold(2.0);
        file.release(1);
        reads.update(1);
      use:
        hold(5.0);
        repeat;
      end***reader***;

      entity class writer;
      begin
      gather:
        hold(5.0);
      write:
        file.acquire(3);
        hold(3.0);
        file.release(3);
        writes.update(1);
        repeat;
      end***writer***;

      outf :- new outfile("o dske:p4");
      outf.open(blanks(70));
      trace;
      reads  :- new count("reads");
      writes :- new count("writes");
      file   :- new res("file", 3);
      new reader("r").schedule(0.0);
      new writer("w").schedule(0.0);
      new reader("r").schedule(0.0);
      new reader("r").schedule(2.0);
      new writer("w").schedule(1.0);
      hold(25.0);
    end;
end;
```

-------------------- program 5 --------------------

```
begin external class demos;
  demos
    begin
      ref(res)file;
      ref(count)reads, writes;

      entity class reader;
      begin
        priority := 1;
      loop: read:
        file.acquire(1);
        hold(2.0);
        file.release(1);
        reads.update(1);
      use:
        hold(5.0);
        repeat;
      end***reader***;

      entity class writer;
      begin
      gather:
        hold(5.0);
      write:
        file.acquire(3);
        hold(3.0);
        file.release(3);
        writes.update(1);
        repeat;
      end***writer***;

      outf :- new outfile("o dske:p5");
      outf.open(blanks(70));
      trace;
      reads  :- new count("reads");
      writes :- new count("writes");
      file   :- new res("file", 3);
      new reader("r").schedule(0.0);
      new writer("w").schedule(0.0);
      new reader("r").schedule(0.0);
      new reader("r").schedule(2.0);
      new writer("w").schedule(1.0);
      hold(25.0);
    end;
end;
```

-------------------- program 6 --------------------

```
begin external class demos;
demos
  begin
    ref(bin)array q(1:2);
    ref(rdist)array next(1:2);
    ref(bin)shutdown;
    ref(rdist)crossing;
    ref(tally)load;
    ref(count)trips, empties;

    entity class ferry;
    begin integer side, c;
      for side := 1, 2 do
      begin
    loading:
        c := 0;
        while c < 6 and q(side).avail > 0 do
        begin q(side).take(1);
          hold(0.5);
          c := c + 1;
        end;
        load.update(c);
        if c = 0 then empties.update(1);
    cross:
        hold(crossing.sample);
    unload:
        hold(c*0.5);
      end;
      trips.update(1);
      if time < 1305.0 then repeat;
      shutdown.give(1);
    end***ferry***;

    entity class arrival(side); integer side;
    begin hold(next(side).sample);
      q(side).give(1);
      repeat;
    end***repeat***;
```

```
    outf :- new outfile("o dske:p6");
    outf.open(blanks(70));
    hold(420.0);
    q(1)     :- new bin("mainland", 3);
    q(2)     :- new bin("island",  1);
    shutdown :- new bin("shutdown", 0);
    next(1)  :- new negexp("mainland", 0.15);
    next(2)  :- new negexp("island",   0.15);
    crossing :- new normal("crossing", 8.0, 0.5);
    trips    :- new count("trips");
    empties  :- new count("empty trips");
    load     :- new tally("av. load");
    new arrival("arr", 1).schedule(0.0);
    new arrival("arr", 2).schedule(0.0);
    new ferry("ferry").schedule(0.0);
    shutdown.take(1);
  end;
end;
```

```
-------------------- program 7 --------------------


begin external class demos;
demos
      begin
        integer k;
        ref(histogram)thru;
        ref(waitq)array requestq(1:6);
        ref(res)array terminal(1:6);
        ref(bin)buffers;
        ref(rdist)arrivals, keyin, process, read;
        ref(idist)terminals;

        entity class scanner;
        begin integer n; boolean b; ref(query)q;
          for n := 1 step 1 until 6 do
          begin
        rotate:
            hold(0.0027);
        test:
            b := requestq(n).length > 0;
            hold(0.0027);
            if b then
            begin
        transfer:
              q :- requestq(n).coopt;
              buffers.take(1);
              hold(0.0117);
              q.schedule(0.0);
            end;
          end;
          repeat;
        end***scanner***;

        entity class query;
        begin integer n; real t;
          new query("query").schedule(arrivals.sample);
          t := time;
          n := terminals.sample;
        joinqanddialrequest:
          terminal(n).acquire(1);
          hold(keyin.sample);
        awaitprocessing:
          requestq(n).wait;
        processing:
          hold(process.sample);
        sendreply:
          hold(0.0397);
          buffers.give(1);
        readreply:
```

```
  hold(read.sample);
  terminal(n).release(1);
  thru.update(time-t);
end***query***;
```

```
        outf :- new outfile("o dske:p7");
        outf.open(blanks(70));
        arrivals  :- new negexp("arr", 5.0);
        terminals :- new randint("terminals", 1, 6);
        keyin     :- new uniform("key in", 0.3, 0.5);
        process   :- new uniform("process", 0.05, 0.10);
        read      :- new uniform("read", 0.6, 0.8);
        thru      :- new histogram("thru times", 1.0, 11.0, 10);

        for k := 1 step 1 until 6 do
        begin
          requestq(k) :- new waitq(edit("request", k));
          terminal(k):- new res(edit("terminal", k), 1);
        end;

        buffers :- new bin("buffers", 3);
        new scanner("scanner").schedule(0.0);
        new query("q").schedule(0.0);
        hold(60.0);
      end;
    end;
```

-------------------- program 8 --------------------

```
begin external class demos;
demos
    begin ref(res)tugs, jetties;
    ref(condq)dockq; boolean lowtide;
    ref(rdist)next, discharge;

    entity class boat;
    begin
      new boat("boat").schedule(next.sample);
    dock:
      jetties.acquire(1);
      dockq.waituntil(tugs.avail >= 2 and not lowtide);
      tugs.acquire(2);
      hold(2.0);
      tugs.release(2);
      dockq.signal;
    unload:
      hold(discharge.sample);
    leave:
      tugs.acquire(1);
      hold(2.0);
      tugs.release(1);  jetties.release(1);
      dockq.signal;
    end***boat***;

    entity class tide;
    begin
    lowtideon:
      lowtide := true;
      hold(4.0);
      lowtide := false;
      dockq.signal;
    lowtideoff:
      hold(9.0);
      repeat;
    end***tide***;

    outf :- new outfile("o dske:p8");
    outf.open(blanks(70));
    trace;
    tugs      :- new res("tugs", 3);
    jetties   :- new res("jetties", 2);
    dockq     :- new condq("dockq");
    next      :- new negexp("next boat", 0.1);
    discharge :- new normal("discharge", 14.0, 3.0);
    new tide("tide").schedule(1.0);
    new boat("boat").schedule(0.0);
    hold(50.0);
```

```
    notrace;
    hold(28.0*24.0-50.0);
  end;
end;
```

-------------------- program 9 --------------------

```
begin external class demos;
demos
  begin integer k;
    ref(res)array fork(1:5);
    ref(idist)think, eat;
    ref(condq)q;

    entity class philosopher(n); integer n;
    begin ref(res)l, r;
      l :- fork(n);
      r :- fork(if n=5 then 1 else n+1);
    loop:
      hold(think.sample);
      q.waituntil(l.avail > 0 and r.avail > 0);
      l.acquire(1);  r.acquire(1);
      hold(eat.sample);
      l.release(1); r.release(1);
      q.signal;
      repeat;
    end***philosopher***;

    outf :- new outfile("o dske:p9");
    outf.open(blanks(70));
    trace;
    q :- new condq("await eat");
    q.all := true;
    think :- new randint("think", 20, 30);
    eat   :- new randint("eat",   10, 20);
    for k := 1 step 1 until 5 do
      fork(k) :- new res(edit("fork", k), 1);
    for k := 1 step 1 until 5 do
      new philosopher("p", k).schedule(0.0);
    hold(180.0);
  end;
end;
```

-------------------- program 10 --------------------

```
begin external class demos;
demos
  begin
    real rate;
    ref(entity)user;  ref(res)hopper;
    ref(idist)p, vol; ref(rdist)next;

    boolean procedure and2(a, b); name a, b; boolean a, b;
      and2 := if a then b else false;

    entity class lorry;
    begin real tleft, load, start;
      new lorry("l").schedule(next.sample);
      priority := p.sample;
      load := 5*vol.sample;
      tleft := load/rate;
      if and2(user =/= none, priority > user.priority) then
         user.interrupt(1);
      while tleft > 0.0 do
      begin hopper.acquire(1);
        user :- current;
        start := time;
        hold(tleft);
      doneorinterrupted:
        user :- none;
        hopper.release(1);
        if interrupted = 0 then tleft := 0.0 else
        begin interrupted := 0.0;
          tleft := tleft - (time-start);
          priority := priority + 1;
        end;
      end;
    end***lorry***;

    outf :- new outfile("o dske:p10");
    outf.open(blanks(70));
    trace;
    rate   := 1.0;
    p      :- new randint("priority", 1, 4);
    vol    :- new randint("lorry load", 1, 3);
    next   :- new negexp("next lorry", 1/12);
    hopper :- new res("hopper", 1);
    new lorry("l").schedule(0.0);
    hold(120.0);
    notrace;
    hold(360.0);
  end;
  end;
```

-------------------- program 11 --------------------

```
begin external class demos;
demos
begin
  ref(sdigger)s;
  ref(condq)q; ref(waitq)ltq, stq;
  ref(rdist)nextl, nexts;

  entity class truck;
  begin
    real load;
  end***truck***;

  truck class ltruck;
  begin
    load := 20.0;
    new ltruck("l").schedule(nextl.sample);
    while load > 0.0 do
    begin q.signal;
      ltq.wait;
    end;
  end***ltruck***;

  truck class struck;
  begin
    load := 5.0;
    new struck("s").schedule(nexts.sample);
    if s.t is ltruck then s.interrupt(1) else q.signal;
    stq.wait;
  end***struck***;

  entity class digger;
  begin
    ref(truck)t; real rate;
  end***digger***;

  digger class ldigger;
  begin
    rate := 240.0;
  loop:
    t :- ltq.coopt;
    hold(t.load/rate);
    t.load := 0.0;
    t.schedule(0.0);
    t :- none;
    if ltq.length = 0 and s.t is ltruck then s.interrupt(2);
    repeat;
  end***ldigger***;
```

```
    digger class sdigger;
    begin real start;
      rate := 60.0;
    loop:
      q.waituntil(stq.length > 0 or ltq.length >
  ltq.masterq.length);
      if stq.length > 0 then
      begin
        t :- stq.coopt;
        hold(t.load/rate);
      end else
      begin
        start := time;
        t :- ltq.coopt;
        hold(t.load/rate);
        if interrupted = 0 then t.load := 0.0 else
        begin t.load := t.load-(time-start)*rate;
          t.priority := 1;
          interrupted := 0;
        end;
      end;
      t.schedule(0.0);
      t :- none;
      repeat;
    end***sdigger***;

    outf :- new outfile("o dske:p11");
    outf.open(blanks(70));
    trace;
    nextl :- new negexp("next large", 22.0);
    nexts :- new negexp("next small", 10.0);
    q     :- new condq("sq");
    stq   :- new waitq("s truck q");
    ltq   :- new waitq("l truck q");
    s     :- new sdigger("s digger");
    s.schedule(0.0);
    new ldigger("l digger").schedule(0.0);
    new ldigger("l digger").schedule(0.0);
    new ltruck("l").schedule(0.0);
    new struck("s").schedule(0.0);
    hold(0.5);
    notrace;
    hold(9.5);
  end;
end;
```

-------------------- program 12 --------------------

```
begin external class demos;
demos
  begin
    ref(waitq)tankq;
    ref(rdist)arr;  ref(idist)size;
    real pumprate, drate, setuptime;

    entity class tanker;
    begin
      ref(shoretank)st;
      integer load;
      new tanker("t").schedule(arr.sample);
      load := 5*size.sample;
    awaitshoretank:
      tankq.find(st, st.free >= load);
      hold(setuptime + load*pumprate);
      st.free := st.free-load;
      st.schedule(0.0);
    end***tanker***;

    entity class shoretank(free); integer free;
    begin
      integer max;
      max := 70;
    loop: reload:
      while free >= 20 do
        tankq.wait;
    discharge:
      hold((max-free)*drate);
      free := max;
      repeat;
    end***shoretank***;

    outf :- new outfile("o dske:p12");
    outf.open(blanks(70));
    setuptime := 0.5;
    pumprate  := 1.0;
    drate     := 0.25;
    arr       :- new negexp("arrivals", 0.125);
    size      :- new randint("size", 3, 5);
    tankq     :- new waitq("shore tanks");
    new shoretank("s", 70).schedule(0.0);
    new shoretank("s", 70).schedule(0.0);
    new shoretank("s", 45).schedule(12.0);
    new shoretank("s", 25).schedule(3.5);
    new shoretank("s", 70).schedule(8.0);
    new tanker("t").schedule(0.0);
    hold(1000.0);
```

```
   end;
end;
```

-------------------- program 13 --------------------

```
begin
  external class demos;
  demos
    begin ref(waitq)bayq;
      ref(rdist)nextlorry, fill, nexttrip;
      ref(res)weighbridge, crane, bays, vanspaces;
      ref(bin)fullhoppers, emptyhoppers;

      entity class van;
      begin
        integer k;
      enter:
        vanspaces.acquire(1);
        weighbridge.acquire(1);
        hold(2.0);
        weighbridge.release(1);
      unload:
        crane.acquire(1);
        for k := 1 step 1 until 3 do
        begin
          emptyhoppers.take(1);
          hold(fill.sample);
          fullhoppers.give(1);
        end;
        crane.release(1);
      leave:
        weighbridge.acquire(1);
        hold(2.0);
        weighbridge.release(1);
        vanspaces.release(1);
      nextload:
        hold(98.0 + nexttrip.sample);
        repeat;
      end***van***;

      entity class lorry;
      begin
        new lorry("lorry").schedule(nextlorry.sample);
      enter:
        bays.acquire(1);
        weighbridge.acquire(1);
        hold(3.0);
        weighbridge.release(1);
      load:
        bayq.wait;

      exit:
        weighbridge.acquire(1);
```

```
  hold(3.0);
  weighbridge.release(1);
  bays.release(1);
end***lorry***;
```

```
        entity class production;
        begin
          ref(lorry)l;
        firsthopper:
          fullhoppers.take(1);
          hold(10.0);
        findtruck:
          l :- bayq.coopt;
          hold(15.0);
          emptyhoppers.give(1);
        secondhopper:
          fullhoppers.take(1);
          hold(25.0);
          emptyhoppers.give(1);
          l.schedule(10.0);
          repeat;
        end***production line***;

        integer k;

        outf :- new outfile("o dske:p13");
        outf.open(blanks(70));
        nextlorry    :- new negexp("next lorry", 0.1);
        fill         :- new normal("fill hopper", 5.0, 1.0);
        nexttrip     :- new negexp("van return", 0.1);
        weighbridge  :- new res("weighbridge", 1);
        crane        :- new res("crane", 1);
        bays         :- new res("bays", 6);
        vanspaces    :- new res("van spaces", 4);
        bayq         :- new waitq("await container");
        fullhoppers  :- new bin("full hoppers",  3);
        emptyhoppers :- new bin("empty hoppers", 5);
        new lorry("l").schedule(0.0);
        for k := 1 step 1 until 7 do
          new van("v").schedule((k-1)*14);
        for k := 1 step 1 until 5 do
          new production("p-line").schedule(0.0);
        hold(480.0);
      end;
    end;
```

-------------------- program 14 --------------------

```
begin
  external class demos;
  demos
    begin
      ref(rdist)burst, next;
      ref(res)cpu;

      entity class program;
      begin
        new program("p").schedule(next.sample);
        cpu.acquire(1);
        hold(burst.sample);
        cpu.release(1);
      end***program***;

      outf :- new outfile("o dske:p14");
      outf.open(blanks(70));
      burst :- new negexp("cpu burst", 1.0);
      next  :- new negexp("next job", 0.875);
      cpu   :- new res("cpu", 1);

      new program("p").schedule(0.0);
      hold(100.0);
    end;
end;
```

-------------------- program 15 --------------------

```
begin
  external class demos;
  demos
    begin
      real slice;
      ref(rdist)burst, next;
      ref(res)cpu, multi;

      entity class program;
      begin
        real tleft, t;

        new program("p").schedule(next.sample);
        multi.acquire(1);
        tleft := burst.sample;
        while tleft > 0.0 do
        begin
          cpu.acquire(1);
          t := if tleft < slice then tleft else slice;
          hold(t);
          tleft := tleft - t;
          cpu.release(1);
        end;
        multi.release(1);
      end***program***;

      outf :- new outfile("o dske:p15");
      outf.open(blanks(70));
      burst :- new negexp("cpu burst", 1.0);
      next  :- new negexp("next job", 0.875);
      cpu   :- new res("cpu", 1);
      multi :- new res("multilevel", 4);
      slice := 0.040;
      new program("p").schedule(0.0);
      hold(100.0);
    end;
end;
```

-------------------- program 16 --------------------

```
begin
  external class demos;
  demos
    begin
      ref(rdist)burst, next, transfer;
      ref(res)cpu, multi, disk;
      ref(idist)ios;

      entity class program;
      begin
        integer k, n;
        multi.acquire(1);
        n := ios.sample//10;
        for k := 1 step 1 until n do
        begin
          cpu.acquire(1);
          hold(burst.sample);
          cpu.release(1);
          disk.acquire(1);
          hold(transfer.sample);
          disk.release(1);
        end;
        multi.release(1);
      end***program***;

      integer k;

      outf :- new outfile("o dske:p16");
      outf.open(blanks(70));
      burst    :- new negexp("cpu burst", 1.0);
      next     :- new negexp("next job", 0.0875);
      transfer :- new uniform("transfer", 0.005, 0.085);
      ios      :- new randint("io's", 25, 75);
      disk     :- new res("disk drive", 2);
      cpu      :- new res("cpu", 1);
      multi    :- new res("multi", 4);

      for k := 1 step 1 until 7 do
        new program("p").schedule(0.0);
      while time < 100.0 do
      begin
        new program("p").schedule(0.0);
        hold(next.sample);
      end;
    end;
end;
```

-------------------- program 17 --------------------

```
begin
  external class demos;
  demos
    begin
      ref(res)cu;
      ref(res)array disk(1:8);
      ref(rdist)next, head, rotate;

      entity class request(n); integer n;
      begin
        new request("r", n).schedule(next.sample);
      passseekinfo:
        disk(n).acquire(1);
        cu.acquire(1);
        hold(0.0);
        cu.release(1);
      movearms:
        hold(head.sample);
      dotransfer:
        cu.acquire(1);
        hold(rotate.sample + transfer);
        cu.release(1);
        disk(n).release(1);
      end***request***;

      integer k;
      real transfer;

      outf :- new outfile("o dske:p17");
      outf.open(blanks(70));
      for k := 1 step 1 until 8 do
        disk(k) :- new res(edit("disk", k), 1);
      cu       :- new res("channel unit", 1);
      next     :- new negexp("next request", 10.0);
      head     :- new uniform("h-movement" , 0.005, 0.085);
      rotate   :- new uniform("rotate time", 0.000, 0.018);
      transfer := 0.001;
      for k := 1 step 1 until 8 do
        new request("r", k).schedule(next.sample*k/8);
      hold(3.0);
    end;
end;
```

-------------------- program 18 --------------------

```
begin external class demos;
demos
  begin
    ref(bin)b;

    entity class p;
    begin
      hold(3.0);
      b.give(1);
      repeat;
    end***p***;

    entity class c;
    begin
      b.take(2);
      hold(2.0);
      repeat;
    end***c***;

    outf :- new outfile("o dske:p18");
    outf.open(blanks(70));
    trace;
    b :- new bin("b", 2);
    new p("p").schedule(0.0);
    new c("c").schedule(6.0);
    hold(25.0);
  end;
end;
```

-------------------- program 19 --------------------

```
begin external class demos;
demos
  begin
    ref(waitq)q;

    entity class p;
    begin ref(entity)e;
      e :- q.coopt;
      hold(2.0);
      e.schedule(time);
      repeat;
    end***p***;

    entity class c;
    begin
      q.wait;
      repeat;
    end***c***;

    q :- new waitq("waiting");
    outf :- new outfile("o dske:p19");
    outf.open(blanks(70));
    trace;
    new p("p").schedule(0.0);
    new p("p").schedule(5.0);
    new c("c").schedule(0.0);
    new c("c").schedule(1.0);
    new c("c").schedule(2.0);
    new c("c").schedule(3.0);
    hold(35.0);
  end;
end;
```

```
-------------------- program 20 --------------------


begin
  external class demos;
  demos
    begin
      ref(rdist)burst, next;
      ref(waitq)cpuq;
      ref(tally)thru;

      entity class program;
      begin
        real t;
        t := time;
        new program("p").schedule(next.sample);
        cpuq.wait;
        thru.update(time-t);
      end***program***;

      entity class cpu;
      begin
        ref(entity)p;
        p :-cpuq.coopt;
        hold(burst.sample);
        p.schedule(0.0);
        repeat;
      end***cpu***;

      outf :- new outfile("o dske:p20");
      outf.open(blanks(70));
      thru  :- new tally("thru times");
      burst :- new negexp("cpu burst", 1.0);
      next  :- new negexp("next job", 0.875);
      cpuq  :- new waitq("cpu q");
      new cpu("cpu").schedule(0.0);
      new program("p").schedule(0.0);
      hold(100.0);
    end;
end;
```

appendix c

... a n d t h e i r o u t p u t s

-------------------- program 0 output --------------------

(empty file)

```
     -------------------- program 1 output --------------------


                       clock time =      0.000
**********************************************************************
*                                                                    *
*                t r a c i n g   c o m m e n c e s                   *
*                                                                    *
**********************************************************************

      time/ current     and its action(s)

     0.000 demos        schedules boat 1 now
                        schedules boat 2 at 1.000
                        schedules boat 3 at 15.000
                        holds for 36.000, until 36.000
           boat 1       seizes 1 of jetties
                        seizes 2 of tugs
                        holds for 2.000, until 2.000
     1.000 boat 2       seizes 1 of jetties
                        awaits 2 of tugs
     2.000 boat 1       releases 2 to tugs
                        holds for 14.000, until 16.000
           boat 2       seizes 2 of tugs
                        holds for 2.000, until 4.000
     4.000              releases 2 to tugs
                        holds for 14.000, until 18.000
    15.000 boat 3       awaits 1 of jetties
    16.000 boat 1       seizes 1 of tugs
                        holds for 2.000, until 18.000
    18.000 boat 2       seizes 1 of tugs
                        holds for 2.000, until 20.000
           boat 1       releases 1 to tugs
                        releases 1 to jetties
                        ***terminates
           boat 3       seizes 1 of jetties
                        seizes 2 of tugs
                        holds for 2.000, until 20.000
    20.000 boat 2       releases 1 to tugs
                        releases 1 to jetties
                        ***terminates
           boat 3       releases 2 to tugs
                        holds for 14.000, until 34.000
    34.000              seizes 1 of tugs
                        holds for 2.000, until 36.000
```

```
                       clock time =     36.000
**********************************************************************
*                                                                    *
*                            r e p o r t                             *
*                                                                    *
**********************************************************************


                            r e s o u r c e s
                            *****************

title        /   (re)set/   obs/ lim/ min/ now/  % usage/ av. wait/qmax
tugs               0.000      5    3    0    2    16.667     0.167    1
jetties            0.000      2    2    0    1    76.389     1.000    1
```

```
     -------------------- program 2 output --------------------

                       clock time =      0.000
**********************************************************************
*                                                                    *
*                t r a c i n g   c o m m e n c e s                   *
*                                                                    *
**********************************************************************

      time/ current     and its action(s)

     0.000 demos        schedules boat 1 now
                        holds for 50.000, until 50.000
           boat 1       schedules boat 2 at 26.574
                        seizes 1 of jetties
                        seizes 2 of tugs
                        holds for 2.000, until 2.000
     2.000              releases 2 to tugs
                        holds for 16.385, until 18.385
    18.385              seizes 1 of tugs
                        holds for 2.000, until 20.385
    20.385              releases 1 to tugs
                        releases 1 to jetties
                        ***terminates
    26.574 boat 2       schedules boat 3 at 33.116
                        seizes 1 of jetties
                        seizes 2 of tugs
                        holds for 2.000, until 28.574
    28.574              releases 2 to tugs
                        holds for 14.724, until 43.298
    33.116 boat 3       schedules boat 4 at 41.836
                        seizes 1 of jetties
                        seizes 2 of tugs
                        holds for 2.000, until 35.116
    35.116              releases 2 to tugs
                        holds for 16.287, until 51.403
    41.836 boat 4       schedules boat 5 at 56.849
                        awaits 1 of jetties
    43.298 boat 2       seizes 1 of tugs
                        holds for 2.000, until 45.298
    45.298              releases 1 to tugs
                        releases 1 to jetties
                        ***terminates
           boat 4       seizes 1 of jetties
                        seizes 2 of tugs
                        holds for 2.000, until 47.298
    47.298              releases 2 to tugs
                        holds for 14.441, until 61.739

                       clock time =     50.000
**********************************************************************
```

```
*                                                                     *
*              t r a c i n g   s w i t c h e d   o f f                *
*                                                                     *
***********************************************************************
```

```
                         clock time =    672.000
**********************************************************************
*                                                                    *
*                             r e p o r t                            *
*                                                                    *
**********************************************************************


                       d i s t r i b u t i o n s
                       *************************

title        /   (re)set/   obs/type     /         a/         b/     seed
next boat           0.000     64 negexp        0.100             33427485
discharge           0.000     58 normal       14.000      3.000  22276755


                           r e s o u r c e s
                           *****************

title        /   (re)set/   obs/ lim/ min/ now/  % usage/ av. wait/qmax
tugs                0.000    114    3    0    3    17.063 2.854e-02    1
jetties             0.000     56    2    0    0    78.566     5.498    6
```

```
     -------------------- program 3 output --------------------

                       clock time =      0.000
**********************************************************************
*                                                                    *
*                t r a c i n g   c o m m e n c e s                   *
*                                                                    *
**********************************************************************

      time/ current     and its action(s)

     0.000 demos        schedules boat 1 now
                        holds for 50.000, until 50.000
           boat 1       schedules boat 2 at 26.574
                        seizes 2 of tugs
                        seizes 1 of jetties
                        holds for 2.000, until 2.000
     2.000              releases 2 to tugs
                        holds for 16.385, until 18.385
    18.385              seizes 1 of tugs
                        holds for 2.000, until 20.385
    20.385              releases 1 to tugs
                        releases 1 to jetties
                        ***terminates
    26.574 boat 2       schedules boat 3 at 33.116
                        seizes 2 of tugs
                        seizes 1 of jetties
                        holds for 2.000, until 28.574
    28.574              releases 2 to tugs
                        holds for 14.724, until 43.298
    33.116 boat 3       schedules boat 4 at 41.836
                        seizes 2 of tugs
                        seizes 1 of jetties
                        holds for 2.000, until 35.116
    35.116              releases 2 to tugs
                        holds for 16.287, until 51.403
    41.836 boat 4       schedules boat 5 at 56.849
                        seizes 2 of tugs
                        awaits 1 of jetties
    43.298 boat 2       seizes 1 of tugs
                        holds for 2.000, until 45.298
    45.298              releases 1 to tugs
                        releases 1 to jetties
                        ***terminates
           boat 4       seizes 1 of jetties
                        holds for 2.000, until 47.298
    47.298              releases 2 to tugs
                        holds for 14.441, until 61.739

                       clock time =     50.000
**********************************************************************
```

```
*                                                                    *
*              t r a c i n g   s w i t c h e d   o f f               *
*                                                                    *
**********************************************************************
```

```
                       clock time =    672.000
**********************************************************************
*                                                                    *
*                             r e p o r t                            *
*                                                                    *
**********************************************************************



                      d i s t r i b u t i o n s
                      *************************

title       /   (re)set/   obs/type     /        a/         b/     seed
next boat         0.000     64 negexp       0.100               33427485
discharge         0.000     10 normal      14.000      3.000  22276755


                          r e s o u r c e s
                          *****************

title       /   (re)set/   obs/ lim/ min/ now/  % usage/ av. wait/qmax
tugs              0.000      18    3    0    1    60.654     0.111   55
jetties           0.000       8    2    0    0    96.787     0.862    1
```

```
     -------------------- program 4 output --------------------

                         clock time =      0.000
**********************************************************************
*                                                                    *
*                t r a c i n g   c o m m e n c e s                   *
*                                                                    *
**********************************************************************


      time/ current     and its action(s)

     0.000 demos        schedules r 1 now
                        schedules w 1 now
                        schedules r 2 now
                        schedules r 3 at 2.000
                        schedules w 2 at 1.000
                        holds for 25.000, until 25.000
           r 1          seizes 1 of file
                        holds for 2.000, until 2.000
           w 1          holds for 5.000, until 5.000
           r 2          seizes 1 of file
                        holds for 2.000, until 2.000
     1.000 w 2          holds for 5.000, until 6.000
     2.000 r 3          seizes 1 of file
                        holds for 2.000, until 4.000
           r 1          releases 1 to file
                        holds for 5.000, until 7.000
           r 2          releases 1 to file
                        holds for 5.000, until 7.000
     4.000 r 3          releases 1 to file
                        holds for 5.000, until 9.000
     5.000 w 1          seizes 3 of file
                        holds for 3.000, until 8.000
     6.000 w 2          awaits 3 of file
     7.000 r 1          awaits 1 of file
           r 2          awaits 1 of file
     8.000 w 1          releases 3 to file
                        holds for 5.000, until 13.000
           w 2          seizes 3 of file
                        holds for 3.000, until 11.000
     9.000 r 3          awaits 1 of file
    11.000 w 2          releases 3 to file
                        holds for 5.000, until 16.000
           r 1          seizes 1 of file
                        holds for 2.000, until 13.000
           r 2          seizes 1 of file
                        holds for 2.000, until 13.000
           r 3          seizes 1 of file
                        holds for 2.000, until 13.000
    13.000 w 1          awaits 3 of file
           r 1          releases 1 to file
```

```
                 holds for 5.000, until 18.000
r 2              releases 1 to file
                 holds for 5.000, until 18.000
r 3              releases 1 to file
```

```
                    holds for 5.000, until 18.000
       w 1          seizes 3 of file
                    holds for 3.000, until 16.000
16.000 w 2          awaits 3 of file
       w 1          releases 3 to file
                    holds for 5.000, until 21.000
       w 2          seizes 3 of file
                    holds for 3.000, until 19.000
18.000 r 1          awaits 1 of file
       r 2          awaits 1 of file
       r 3          awaits 1 of file
19.000 w 2          releases 3 to file
                    holds for 5.000, until 24.000
       r 1          seizes 1 of file
                    holds for 2.000, until 21.000
       r 2          seizes 1 of file
                    holds for 2.000, until 21.000
       r 3          seizes 1 of file
                    holds for 2.000, until 21.000
21.000 w 1          awaits 3 of file
       r 1          releases 1 to file
                    holds for 5.000, until 26.000
       r 2          releases 1 to file
                    holds for 5.000, until 26.000
       r 3          releases 1 to file
                    holds for 5.000, until 26.000
       w 1          seizes 3 of file
                    holds for 3.000, until 24.000
24.000 w 2          awaits 3 of file
       w 1          releases 3 to file
                    holds for 5.000, until 29.000
       w 2          seizes 3 of file
                    holds for 3.000, until 27.000


                   clock time =     25.000
**********************************************************************
*                                                                    *
*                             r e p o r t                            *
*                                                                    *
**********************************************************************


                             c o u n t s
                             ***********

                 title       /   (re)set/   obs
                 reads             0.000      9
                 writes            0.000      5
```

r e s o u r c e s
****

title / (re)set/ obs/ lim/ min/ now/ % usage/ av. wait/qmax

file	0.000	14	3	0	0	88.000	1.000	3

```
-------------------- program 5 output --------------------


                   clock time =      0.000
**********************************************************************
*                                                                    *
*                 t r a c i n g   c o m m e n c e s                  *
*                                                                    *
**********************************************************************

      time/ current     and its action(s)

     0.000 demos        schedules r 1 now
                        schedules w 1 now
                        schedules r 2 now
                        schedules r 3 at 2.000
                        schedules w 2 at 1.000
                        holds for 25.000, until 25.000
           r 1          seizes 1 of file
                        holds for 2.000, until 2.000
           w 1          holds for 5.000, until 5.000
           r 2          seizes 1 of file
                        holds for 2.000, until 2.000
     1.000 w 2          holds for 5.000, until 6.000
     2.000 r 3          seizes 1 of file
                        holds for 2.000, until 4.000
           r 1          releases 1 to file
                        holds for 5.000, until 7.000
           r 2          releases 1 to file
                        holds for 5.000, until 7.000
     4.000 r 3          releases 1 to file
                        holds for 5.000, until 9.000
     5.000 w 1          seizes 3 of file
                        holds for 3.000, until 8.000
     6.000 w 2          awaits 3 of file
     7.000 r 1          awaits 1 of file
           r 2          awaits 1 of file
     8.000 w 1          releases 3 to file
                        holds for 5.000, until 13.000
           r 1          seizes 1 of file
                        holds for 2.000, until 10.000
           r 2          seizes 1 of file
                        holds for 2.000, until 10.000
     9.000 r 3          seizes 1 of file
                        holds for 2.000, until 11.000
    10.000 r 1          releases 1 to file
                        holds for 5.000, until 15.000
           r 2          releases 1 to file
                        holds for 5.000, until 15.000
    11.000 r 3          releases 1 to file
                        holds for 5.000, until 16.000
```

```
       w 2          seizes 3 of file
                    holds for 3.000, until 14.000
13.000 w 1          awaits 3 of file
14.000 w 2          releases 3 to file
```

```
                    holds for 5.000, until 19.000
       w 1          seizes 3 of file
                    holds for 3.000, until 17.000
15.000 r 1          awaits 1 of file
       r 2          awaits 1 of file
16.000 r 3          awaits 1 of file
17.000 w 1          releases 3 to file
                    holds for 5.000, until 22.000
       r 1          seizes 1 of file
                    holds for 2.000, until 19.000
       r 2          seizes 1 of file
                    holds for 2.000, until 19.000
       r 3          seizes 1 of file
                    holds for 2.000, until 19.000
19.000 w 2          awaits 3 of file
       r 1          releases 1 to file
                    holds for 5.000, until 24.000
       r 2          releases 1 to file
                    holds for 5.000, until 24.000
       r 3          releases 1 to file
                    holds for 5.000, until 24.000
       w 2          seizes 3 of file
                    holds for 3.000, until 22.000
22.000 w 1          awaits 3 of file
       w 2          releases 3 to file
                    holds for 5.000, until 27.000
       w 1          seizes 3 of file
                    holds for 3.000, until 25.000
24.000 r 1          awaits 1 of file
       r 2          awaits 1 of file
       r 3          awaits 1 of file


                   clock time =     25.000
**********************************************************************
*                                                                    *
*                            r e p o r t                             *
*                                                                    *
**********************************************************************



                            c o u n t s
                            ***********

                   title       /   (re)set/   obs
                   reads              0.000     9
                   writes             0.000     4


                          r e s o u r c e s
```

****

title /	(re)set/	obs/	lim/	min/	now/	% usage/	av. wait/	qmax
file	0.000	13	3	0	0	84.000	0.929	3

```
    -------------------- program 6 output --------------------


                      clock time =   1315.296
**********************************************************************
*                                                                    *
*                            r e p o r t                             *
*                                                                    *
**********************************************************************



                     d i s t r i b u t i o n s
                     *************************

title       /   (re)set/   obs/type     /        a/         b/     seed
mainland        420.000    133 negexp       0.150            33427485
island          420.000    146 negexp       0.150            22276755
crossing        420.000     78 normal       8.000     0.500  46847980


                            c o u n t s
                            ***********

                    title       /   (re)set/   obs
                    trips           420.000     39
                    empty trips     420.000      2


                           t a l l i e s
                           *************

title       /   (re)set/   obs/  average/est.st.dv/  minimum/  maximum
av. load        420.000     78     3.487     1.634     0.000     6.000


                              b i n s
                              *******

title       /   (re)set/   obs/init/ max/ now/ av. free/ av. wait/qmax
mainland        420.000    132    3    6    2     1.678     0.000    1
island          420.000    145    1    7    7     1.705     0.000    1
shutdown        420.000      1    0    1    0     0.000   895.296    1
```

```
     -------------------- program 7 output --------------------


                       clock time =     60.000
**********************************************************************
*                                                                    *
*                             r e p o r t                            *
*                                                                    *
**********************************************************************



                      d i s t r i b u t i o n s
                      *************************

title       /   (re)set/   obs/type     /        a/         b/     seed
arr               0.000    300 negexp       5.000              33427485
terminals         0.000    300 randint          1          6   22276755
key in            0.000    260 uniform      0.300      0.500   46847980
process           0.000    257 uniform  5.000e-02      0.100   43859043
read              0.000    257 uniform      0.600      0.800   64042082


                         h i s t o g r a m s
                         *******************

                            s u m m a r y

title       /   (re)set/   obs/  average/est.st.dv/  minimum/  maximum
thru times        0.000    254     4.392     3.347     1.080    17.292

cell/lower lim/    n/   freq/  cum %
                                       i-------------------------------
   0 -infinity     0    0.00    0.00   i
   1     1.000    73    0.29   28.74   i*******************************
   2     2.000    44    0.17   46.06   i******************
   3     3.000    29    0.11   57.48   i************
   4     4.000    25    0.10   67.32   i**********
   5     5.000    21    0.08   75.59   i*********
   6     6.000    15    0.06   81.50   i******
   7     7.000    11    0.04   85.83   i*****
   8     8.000     7    0.03   88.58   i***
   9     9.000    11    0.04   92.91   i*****
  10    10.000     3    0.01   94.09   i*
  11    11.000    15    0.06  100.00   i******
                                       i-------------------------------
```

r e s o u r c e s

title	/	(re)set/	obs/	lim/	min/	now/	% usage/	av. wait/	qmax
terminal 1		0.000	40	1	0	0	83.549	4.323	15
terminal 2		0.000	45	1	0	0	96.128	6.677	18
terminal 3		0.000	45	1	0	0	94.530	1.851	4
terminal 4		0.000	42	1	0	0	89.087	2.133	6
terminal 5		0.000	38	1	0	0	78.368	0.652	3
terminal 6		0.000	44	1	0	0	89.604	3.323	7

b i n s

title	/	(re)set/	obs/	init/	max/	now/	av. free/	av. wait/	qmax
buffers		0.000	257	3	3	3	2.457	0.000	1

w a i t q u e u e s

title	/	(re)set/	obs/	qmax/	qnow/	q average/	zeros/	av. wait
request 1		0.000	41	1	0	0.000	41	0.000
request 1	*	0.000	41	1	0	1.017e-02	0	1.488e-02
request 2		0.000	46	1	0	0.000	46	0.000
request 2	*	0.000	46	1	0	1.672e-02	0	2.181e-02
request 3		0.000	45	1	0	0.000	45	0.000
request 3	*	0.000	45	1	0	1.711e-02	0	2.281e-02
request 4		0.000	43	1	0	0.000	43	0.000
request 4	*	0.000	43	1	0	1.276e-02	0	1.780e-02
request 5		0.000	38	1	0	0.000	38	0.000
request 5	*	0.000	38	1	0	1.226e-02	0	1.936e-02
request 6		0.000	44	1	0	0.000	44	0.000
request 6	*	0.000	44	1	0	1.368e-02	0	1.866e-02

```
-------------------- program 8 output --------------------

                     clock time =      0.000
**********************************************************************
*                                                                    *
*                t r a c i n g   c o m m e n c e s                   *
*                                                                    *
**********************************************************************

     time/ current     and its action(s)

    0.000 demos        schedules tide 1 at 1.000
                       schedules boat 1 now
                       holds for 50.000, until 50.000
          boat 1       schedules boat 2 at 26.574
                       seizes 1 of jetties
                       seizes 2 of tugs
                       holds for 2.000, until 2.000
    1.000 tide 1       holds for 4.000, until 5.000
    2.000 boat 1       releases 2 to tugs
                       signals dockq
                       holds for 16.385, until 18.385
    5.000 tide 1       signals dockq
                       holds for 9.000, until 14.000
   14.000              holds for 4.000, until 18.000
   18.000              signals dockq
                       holds for 9.000, until 27.000
   18.385 boat 1       seizes 1 of tugs
                       holds for 2.000, until 20.385
   20.385              releases 1 to tugs
                       releases 1 to jetties
                       signals dockq
                       ***terminates
   26.574 boat 2       schedules boat 3 at 33.116
                       seizes 1 of jetties
                       seizes 2 of tugs
                       holds for 2.000, until 28.574
   27.000 tide 1       holds for 4.000, until 31.000
   28.574 boat 2       releases 2 to tugs
                       signals dockq
                       holds for 14.724, until 43.298
   31.000 tide 1       signals dockq
                       holds for 9.000, until 40.000
   33.116 boat 3       schedules boat 4 at 41.836
                       seizes 1 of jetties
                       seizes 2 of tugs
                       holds for 2.000, until 35.116
   35.116              releases 2 to tugs
                       signals dockq
                       holds for 16.287, until 51.403
```

```
40.000 tide 1       holds for 4.000, until 44.000
41.836 boat 4       schedules boat 5 at 56.849
                    awaits 1 of jetties
43.298 boat 2       seizes 1 of tugs
```

```
                        holds for 2.000, until 45.298
   44.000 tide 1        signals dockq
                        holds for 9.000, until 53.000
   45.298 boat 2        releases 1 to tugs
                        releases 1 to jetties
                        signals dockq
                        ***terminates
          boat 4        seizes 1 of jetties
                        seizes 2 of tugs
                        holds for 2.000, until 47.298
   47.298               releases 2 to tugs
                        signals dockq
                        holds for 14.441, until 61.739


                       clock time =     50.000
**********************************************************************
*                                                                    *
*              t r a c i n g   s w i t c h e d   o f f               *
*                                                                    *
**********************************************************************



                       clock time =    672.000
**********************************************************************
*                                                                    *
*                            r e p o r t                             *
*                                                                    *
**********************************************************************



                      d i s t r i b u t i o n s
                      *************************

title       /   (re)set/   obs/type     /        a/         b/     seed
next boat         0.000     64 negexp       0.100              33427485
discharge         0.000     58 normal      14.000     3.000  22276755


                          r e s o u r c e s
                          *****************

title       /   (re)set/   obs/ lim/ min/ now/  % usage/ av. wait/qmax
tugs              0.000    114    3    0    3    17.063     0.000    1
jetties           0.000     56    2    0    0    81.396     6.343    7


                    c o n d i t i o n   q u e u e s
                    *******************************
```

title /	(re)set/	obs/	qmax/	qnow/	q average/	zeros/	av. wait
dockq	0.000	58	2	0	7.683e-02	36	0.890

```
     -------------------- program 9 output --------------------


                            clock time =      0.000
**********************************************************************
*                                                                    *
*                 t r a c i n g   c o m m e n c e s                  *
*                                                                    *
**********************************************************************

      time/ current     and its action(s)

     0.000 demos        schedules p 1 now
                        schedules p 2 now
                        schedules p 3 now
                        schedules p 4 now
                        schedules p 5 now
                        holds for 180.000, until 180.000
           p 1          holds for 20.000, until 20.000
           p 2          holds for 25.000, until 25.000
           p 3          holds for 24.000, until 24.000
           p 4          holds for 22.000, until 22.000
           p 5          holds for 26.000, until 26.000
    20.000 p 1          seizes 1 of fork 1
                        seizes 1 of fork 2
                        holds for 17.000, until 37.000
    22.000 p 4          seizes 1 of fork 4
                        seizes 1 of fork 5
                        holds for 12.000, until 34.000
    24.000 p 3          w'until in await eat
    25.000 p 2          w'until in await eat
    26.000 p 5          w'until in await eat
    34.000 p 4          releases 1 to fork 4
                        releases 1 to fork 5
                        signals await eat
                        holds for 23.000, until 57.000
           p 3          leaves await eat
                        seizes 1 of fork 3
                        seizes 1 of fork 4
                        holds for 18.000, until 52.000
    37.000 p 1          releases 1 to fork 1
                        releases 1 to fork 2
                        signals await eat
                        holds for 27.000, until 64.000
           p 5          leaves await eat
                        seizes 1 of fork 5
                        seizes 1 of fork 1
                        holds for 12.000, until 49.000
    49.000              releases 1 to fork 5
                        releases 1 to fork 1
                        signals await eat
```

```
                    holds for 23.000, until 72.000
52.000 p 3          releases 1 to fork 3
                    releases 1 to fork 4
                    signals await eat
```

```
                    holds for 25.000, until 77.000
         p 2        leaves await eat
                    seizes 1 of fork 2
                    seizes 1 of fork 3
                    holds for 15.000, until 67.000
  57.000 p 4        seizes 1 of fork 4
                    seizes 1 of fork 5
                    holds for 16.000, until 73.000
  64.000 p 1        w'until in await eat
  67.000 p 2        releases 1 to fork 2
                    releases 1 to fork 3
                    signals await eat
                    holds for 30.000, until 97.000
         p 1        leaves await eat
                    seizes 1 of fork 1
                    seizes 1 of fork 2
                    holds for 18.000, until 85.000
  72.000 p 5        w'until in await eat
  73.000 p 4        releases 1 to fork 4
                    releases 1 to fork 5
                    signals await eat
                    holds for 25.000, until 98.000
  77.000 p 3        seizes 1 of fork 3
                    seizes 1 of fork 4
                    holds for 17.000, until 94.000
  85.000 p 1        releases 1 to fork 1
                    releases 1 to fork 2
                    signals await eat
                    holds for 30.000, until 115.000
         p 5        leaves await eat
                    seizes 1 of fork 5
                    seizes 1 of fork 1
                    holds for 15.000, until 100.000
  94.000 p 3        releases 1 to fork 3
                    releases 1 to fork 4
                    signals await eat
                    holds for 23.000, until 117.000
  97.000 p 2        seizes 1 of fork 2
                    seizes 1 of fork 3
                    holds for 19.000, until 116.000
  98.000 p 4        w'until in await eat
 100.000 p 5        releases 1 to fork 5
                    releases 1 to fork 1
                    signals await eat
                    holds for 25.000, until 125.000
         p 4        leaves await eat
                    seizes 1 of fork 4
                    seizes 1 of fork 5
                    holds for 13.000, until 113.000
 113.000            releases 1 to fork 4
                    releases 1 to fork 5
```

```
                     signals await eat
                     holds for 22.000, until 135.000
115.000 p 1          w'until in await eat
116.000 p 2          releases 1 to fork 2
```

```
                     releases 1 to fork 3
                     signals await eat
                     holds for 21.000, until 137.000
         p 1         leaves await eat
                     seizes 1 of fork 1
                     seizes 1 of fork 2
                     holds for 11.000, until 127.000
 117.000 p 3         seizes 1 of fork 3
                     seizes 1 of fork 4
                     holds for 13.000, until 130.000
 125.000 p 5         w'until in await eat
 127.000 p 1         releases 1 to fork 1
                     releases 1 to fork 2
                     signals await eat
                     holds for 25.000, until 152.000
         p 5         leaves await eat
                     seizes 1 of fork 5
                     seizes 1 of fork 1
                     holds for 18.000, until 145.000
 130.000 p 3         releases 1 to fork 3
                     releases 1 to fork 4
                     signals await eat
                     holds for 25.000, until 155.000
 135.000 p 4         w'until in await eat
 137.000 p 2         seizes 1 of fork 2
                     seizes 1 of fork 3
                     holds for 18.000, until 155.000
 145.000 p 5         releases 1 to fork 5
                     releases 1 to fork 1
                     signals await eat
                     holds for 23.000, until 168.000
         p 4         leaves await eat
                     seizes 1 of fork 4
                     seizes 1 of fork 5
                     holds for 14.000, until 159.000
 152.000 p 1         w'until in await eat
 155.000 p 3         w'until in await eat
         p 2         releases 1 to fork 2
                     releases 1 to fork 3
                     signals await eat
                     holds for 22.000, until 177.000
         p 1         leaves await eat
                     seizes 1 of fork 1
                     seizes 1 of fork 2
                     holds for 10.000, until 165.000
 159.000 p 4         releases 1 to fork 4
                     releases 1 to fork 5
                     signals await eat
                     holds for 21.000, until 180.000
         p 3         leaves await eat
                     seizes 1 of fork 3
```

```
                    seizes 1 of fork 4
                    holds for 20.000, until 179.000
165.000 p 1         releases 1 to fork 1
                    releases 1 to fork 2
```

```
                     signals await eat
                     holds for 22.000, until 187.000
 168.000 p 5         seizes 1 of fork 5
                     seizes 1 of fork 1
                     holds for 16.000, until 184.000
 177.000 p 2         w'until in await eat
 179.000 p 3         releases 1 to fork 3
                     releases 1 to fork 4
                     signals await eat
                     holds for 24.000, until 203.000
         p 2         leaves await eat
                     seizes 1 of fork 2
                     seizes 1 of fork 3
                     holds for 19.000, until 198.000


                    clock time =    180.000
**********************************************************************
*                                                                    *
*                           r e p o r t                              *
*                                                                    *
**********************************************************************



                   d i s t r i b u t i o n s
                   *************************

title       /   (re)set/   obs/type     /        a/        b/     seed
think             0.000     23 randint          20        30  33427485
eat               0.000     20 randint          10        20  22276755


                       r e s o u r c e s
                       *****************

title       /   (re)set/   obs/ lim/ min/ now/  % usage/ av. wait/qmax
fork 1            0.000      7    1    0    0    62.778     0.000    1
fork 2            0.000      7    1    0    0    60.556     0.000    1
fork 3            0.000      7    1    0    0    67.222     0.000    1
fork 4            0.000      8    1    0    1    68.333     0.000    1
fork 5            0.000      7    1    0    0    62.222     0.000    1


                c o n d i t i o n   q u e u e s
                *******************************

title       /   (re)set/   obs/ qmax/ qnow/ q average/zeros/  av. wait
await eat   *     0.000     20     3     0      0.489     8      4.400
```

```
     -------------------- program 10 output --------------------

                      clock time =      0.000
**********************************************************************
*                                                                    *
*                 t r a c i n g   c o m m e n c e s                  *
*                                                                    *
**********************************************************************

      time/ current     and its action(s)

     0.000 demos        schedules l 1 now
                        holds for 120.000, until 120.000
           l 1          schedules l 2 at 7.358
                        seizes 1 of hopper
                        holds for 15.000, until 15.000
     7.358 l 2          schedules l 3 at 88.690
                        interrupts l 1, with n = 1
                        cancels l 1
                        awaits 1 of hopper
           l 1          releases 1 to hopper
                        awaits 1 of hopper
           l 2          seizes 1 of hopper
                        holds for 5.000, until 12.358
    12.358              releases 1 to hopper
                        ***terminates
           l 1          seizes 1 of hopper
                        holds for 7.642, until 20.000
    20.000              releases 1 to hopper
                        ***terminates
    88.690 l 3          schedules l 4 at 101.975
                        seizes 1 of hopper
                        holds for 15.000, until 103.690
   101.975 l 4          schedules l 5 at 107.870
                        awaits 1 of hopper
   103.690 l 3          releases 1 to hopper
                        ***terminates
           l 4          seizes 1 of hopper
                        holds for 5.000, until 108.690
   107.870 l 5          schedules l 6 at 117.565
                        interrupts l 4, with n = 1
                        cancels l 4
                        awaits 1 of hopper
           l 4          releases 1 to hopper
                        awaits 1 of hopper
           l 5          seizes 1 of hopper
                        holds for 10.000, until 117.870
   117.565 l 6          schedules l 7 at 120.236
                        awaits 1 of hopper
   117.870 l 5          releases 1 to hopper
```

```
                    ***terminates
        l 4         seizes 1 of hopper
                    holds for 0.820, until 118.690
118.690             releases 1 to hopper
```

```
                   ***terminates
         l 6         seizes 1 of hopper
                     holds for 10.000, until 128.690


                   clock time =    120.000
**********************************************************************
*                                                                    *
*             t r a c i n g   s w i t c h e d   o f f                *
*                                                                    *
**********************************************************************



                   clock time =    480.000
**********************************************************************
*                                                                    *
*                          r e p o r t                               *
*                                                                    *
**********************************************************************


                     d i s t r i b u t i o n s
                     *************************

title       /   (re)set/   obs/type     /        a/         b/     seed
priority          0.000     44 randint           1          4  33427485
lorry load        0.000     44 randint           1          3  22276755
next lorry        0.000     44 negexp    8.333e-02             46847980


                         r e s o u r c e s
                         *****************

title       /   (re)set/   obs/ lim/ min/ now/  % usage/ av. wait/qmax
hopper            0.000     51    1    0    0    82.252    17.978    8
```

```
     -------------------- program 11 output --------------------


                          clock time =      0.000
**********************************************************************
*                                                                    *
*                  t r a c i n g   c o m m e n c e s                 *
*                                                                    *
**********************************************************************

      time/ current     and its action(s)

     0.000 demos        schedules s digger 1 now
                        schedules l digger 1 now
                        schedules l digger 2 now
                        schedules l 1 now
                        schedules s 1 now
                        holds for 0.500, until 0.500
           s digger 1   w'until in sq
           l digger 1   waits in l truck q
           l digger 2   waits in l truck q
           l 1          schedules l 2 at 0.121
                        signals sq
                        waits in l truck q
           s 1          schedules s 2 at 0.035
                        signals sq
                        waits in s truck q
           s digger 1   leaves sq
                        coopts s 1 from s truck q
                        holds for 0.083, until 0.083
           l digger 1   coopts l 1 from l truck q
                        holds for 0.083, until 0.083
 3.453e-02 s 2          schedules s 3 at 0.194
                        signals sq
                        waits in s truck q
 8.333e-02 s digger 1   schedules s 1 now
                        coopts s 2 from s truck q
                        holds for 0.083, until 0.167
           l digger 1   schedules l 1 now
                        waits in l truck q
           s 1          ***terminates
           l 1          ***terminates
     0.121 l 2          schedules l 3 at 0.151
                        signals sq
                        waits in l truck q
           l digger 2   coopts l 2 from l truck q
                        holds for 0.083, until 0.204
     0.151 l 3          schedules l 4 at 0.190
                        signals sq
                        waits in l truck q
           l digger 1   coopts l 3 from l truck q
```

```
                   holds for 0.083, until 0.234
0.167 s digger 1   schedules s 2 now
                   w'until in sq
      s 2          ***terminates
```

```
0.190 l 4           schedules l 5 at 0.258
                    signals sq
                    waits in l truck q
      s digger 1    leaves sq
                    coopts l 4 from l truck q
                    holds for 0.333, until 0.523
0.194 s 3           schedules s 4 at 0.224
                    interrupts s digger 1, with n = 1
                    cancels s digger 1
                    waits in s truck q
      s digger 1    schedules l 4 now
                    coopts s 3 from s truck q
                    holds for 0.083, until 0.277
      l 4           signals sq
                    waits in l truck q
0.204 l digger 2    schedules l 2 now
                    coopts l 4 from l truck q
                    holds for 0.082, until 0.287
      l 2           ***terminates
0.224 s 4           schedules s 5 at 0.376
                    signals sq
                    waits in s truck q
0.234 l digger 1    schedules l 3 now
                    waits in l truck q
      l 3           ***terminates
0.258 l 5           schedules l 6 at 0.281
                    signals sq
                    waits in l truck q
      l digger 1    coopts l 5 from l truck q
                    holds for 0.083, until 0.342
0.277 s digger 1    schedules s 3 now
                    coopts s 4 from s truck q
                    holds for 0.083, until 0.361
      s 3           ***terminates
0.281 l 6           schedules l 7 at 0.334
                    signals sq
                    waits in l truck q
0.287 l digger 2    schedules l 4 now
                    coopts l 6 from l truck q
                    holds for 0.083, until 0.370
      l 4           ***terminates
0.334 l 7           schedules l 8 at 0.354
                    signals sq
                    waits in l truck q
0.342 l digger 1    schedules l 5 now
                    coopts l 7 from l truck q
                    holds for 0.083, until 0.425
      l 5           ***terminates
0.354 l 8           schedules l 9 at 0.400
                    signals sq
                    waits in l truck q
```

```
0.361 s digger 1   schedules s 4 now
                   coopts l 8 from l truck q
                   holds for 0.333, until 0.694
      s 4          ***terminates
```

```
    0.370 l digger 2   schedules l 6 now
                       interrupts s digger 1, with n = 2
                       cancels s digger 1
                       waits in l truck q
          l 6          ***terminates
          s digger 1   schedules l 8 now
                       w'until in sq
          l 8          signals sq
                       waits in l truck q
          l digger 2   coopts l 8 from l truck q
                       holds for 0.081, until 0.451
    0.376 s 5          schedules s 6 at 0.452
                       signals sq
                       waits in s truck q
          s digger 1   leaves sq
                       coopts s 5 from s truck q
                       holds for 0.083, until 0.459
    0.400 l 9          schedules l10 at 0.429
                       signals sq
                       waits in l truck q
    0.425 l digger 1   schedules l 7 now
                       coopts l 9 from l truck q
                       holds for 0.083, until 0.508
          l 7          ***terminates
    0.429 l10          schedules l11 at 0.432
                       signals sq
                       waits in l truck q
    0.432 l11          schedules l12 at 0.464
                       signals sq
                       waits in l truck q
    0.451 l digger 2   schedules l 8 now
                       coopts l10 from l truck q
                       holds for 0.083, until 0.534
          l 8          ***terminates
    0.452 s 6          schedules s 7 at 0.506
                       signals sq
                       waits in s truck q
    0.459 s digger 1   schedules s 5 now
                       coopts s 6 from s truck q
                       holds for 0.083, until 0.542
          s 5          ***terminates
    0.464 l12          schedules l13 at 0.466
                       signals sq
                       waits in l truck q
    0.466 l13          schedules l14 at 0.524
                       signals sq
                       waits in l truck q


                     clock time =      0.500
**********************************************************************
```

```
*                                                                     *
*               t r a c i n g   s w i t c h e d   o f f               *
*                                                                     *
***********************************************************************
```

```
                       clock time =     10.000
**********************************************************************
*                                                                    *
*                             r e p o r t                            *
*                                                                    *
**********************************************************************



                      d i s t r i b u t i o n s
                      *************************

title        /   (re)set/   obs/type     /        a/         b/     seed
next large         0.000    215 negexp       22.000            33427485
next small         0.000    100 negexp       10.000            22276755


                        w a i t   q u e u e s
                        *********************

title        /   (re)set/   obs/ qmax/ qnow/ q average/zeros/  av. wait
s truck q          0.000     97     1     0      0.000    97      0.000
s truck q    *     0.000     97     4     3      0.796    25  8.082e-02

l truck q          0.000    224     2     0      0.319   185  1.422e-02
l truck q    *     0.000    224    13     8      3.296    48      0.136



                   c o n d i t i o n   q u e u e s
                   *******************************

title        /   (re)set/   obs/ qmax/ qnow/ q average/zeros/  av. wait
sq                 0.000    114     1     0  5.038e-02    99  4.419e-03
```

-------------------- program 12 output --------------------

```
                      clock time =   1000.000
**********************************************************************
*                                                                    *
*                            r e p o r t                             *
*                                                                    *
**********************************************************************


                      d i s t r i b u t i o n s
                      *************************

title        /   (re)set/   obs/type     /        a/        b/     seed
arrivals           0.000    128 negexp       0.125             33427485
size               0.000    128 randint          3         5   22276755


                        w a i t   q u e u e s
                        *********************

title        /   (re)set/   obs/ qmax/ qnow/ q average/zeros/  av. wait
shore tanks        0.000    128     5     0      0.181    99      1.417
shore tanks *      0.000    128     5     2      1.849    30     14.230
```

```
     -------------------- program 13 output --------------------


                       clock time =    480.000
**********************************************************************
*                                                                    *
*                            r e p o r t                             *
*                                                                    *
**********************************************************************



                      d i s t r i b u t i o n s
                      *************************

title       /   (re)set/   obs/type     /        a/         b/     seed
next lorry         0.000     41 negexp       0.100              33427485
fill hopper        0.000     74 normal       5.000     1.000  22276755
van return         0.000     24 negexp       0.100              46847980


                          r e s o u r c e s
                          *****************

title       /   (re)set/   obs/ lim/ min/ now/  % usage/ av. wait/qmax
weighbridge        0.000    121    1    0    1    65.417     1.168    4
crane              0.000     24    1    0    0    81.393     8.943    2
bays               0.000     33    6    0    0    88.570    12.900    4
van spaces         0.000     24    4    0    3    40.525     0.000    1


                               b i n s
                               *******

title       /   (re)set/   obs/init/ max/ now/ av. free/ av. wait/qmax
full hoppers       0.000     73    3    3    0     0.516     5.477    5
empty hopper       0.000     71    5    8    2     2.571     0.219    1


                        w a i t   q u e u e s
                        *********************

title       /   (re)set/   obs/ qmax/ qnow/ q average/zeros/  av. wait
await contai       0.000     38     4     0      0.315    27      3.977
await contai*      0.000     38     3     1      0.578    12      6.743
```

```
-------------------- program 14 output --------------------

                       clock time =    100.000
**********************************************************************
*                                                                    *
*                             r e p o r t                            *
*                                                                    *
**********************************************************************



                      d i s t r i b u t i o n s
                      *************************

title        /   (re)set/   obs/type     /        a/        b/     seed
cpu burst          0.000     88 negexp       1.000            33427485
next job           0.000     88 negexp       0.875            22276755


                          r e s o u r c e s
                          *****************

title        /   (re)set/   obs/ lim/ min/ now/  % usage/ av. wait/qmax
cpu                0.000     88    1    0    1    90.108     3.672   10
```

-------------------- program 15 output --------------------

```
                       clock time =    100.000
**********************************************************************
*                                                                    *
*                             r e p o r t                            *
*                                                                    *
**********************************************************************


                      d i s t r i b u t i o n s
                      *************************

title        /   (re)set/   obs/type     /        a/         b/     seed
cpu burst          0.000     88 negexp       1.000              33427485
next job           0.000     88 negexp       0.875              22276755


                          r e s o u r c e s
                          *****************

title        /   (re)set/   obs/ lim/ min/ now/  % usage/ av. wait/qmax
cpu                0.000   2300    1    0    1    90.108 8.390e-02    4
multilevel         0.000     88    4    0    4    70.769     1.579    6
```

```
-------------------- program 16 output --------------------

                      clock time =    114.768
**********************************************************************
*                                                                    *
*                            r e p o r t                             *
*                                                                    *
**********************************************************************


                      d i s t r i b u t i o n s
                      *************************

title       /   (re)set/   obs/type     /        a/        b/     seed
cpu burst         0.000     88 negexp       1.000               33427485
next job          0.000     12 negexp    8.750e-02              22276755
transfer          0.000     88 uniform   5.000e-03 8.500e-02  46847980
io's              0.000     19 randint          25        75  43859043

                          r e s o u r c e s
                          *****************

title       /   (re)set/   obs/ lim/ min/ now/  % usage/ av. wait/qmax
disk drive        0.000     88    2    0    2     1.665     0.000    1
cpu               0.000     88    1    0    1    78.513     2.669    3
multi             0.000     19    4    0    4    71.624     7.602    5
```

```
-------------------- program 17 output --------------------

                         clock time =      3.000
**********************************************************************
*                                                                    *
*                             r e p o r t                            *
*                                                                    *
**********************************************************************



                        d i s t r i b u t i o n s
                        *************************

title       /   (re)set/   obs/type     /        a/        b/     seed
next request      0.000    249 negexp      10.000              33427485
h-movement        0.000    231 uniform   5.000e-03 8.500e-02  22276755
rotate time       0.000    224 uniform       0.000 1.800e-02  46847980


                            r e s o u r c e s
                            *****************

title       /   (re)set/   obs/ lim/ min/ now/  % usage/ av. wait/qmax
disk 1            0.000     28    1    0    0    71.704 4.998e-02    3
disk 2            0.000     30    1    0    0    74.262 5.787e-02    5
disk 3            0.000     28    1    0    0    70.885     0.104    5
disk 4            0.000     16    1    0    1    48.032 4.707e-03    1
disk 5            0.000     29    1    0    0    72.555     0.101    5
disk 6            0.000     37    1    0    0    94.048     0.184    6
disk 7            0.000     31    1    0    0    66.761 3.130e-02    2
disk 8            0.000     25    1    0    0    59.294 4.319e-02    3
channel unit      0.000    455    1    0    1    72.938 1.030e-02    7
```

```
-------------------- program 18 output --------------------


                       clock time =      0.000
**********************************************************************
*                                                                    *
*                  t r a c i n g   c o m m e n c e s                 *
*                                                                    *
**********************************************************************

      time/ current     and its action(s)

     0.000 demos        schedules p 1 now
                        schedules c 1 at 6.000
                        holds for 25.000, until 25.000
           p 1          holds for 3.000, until 3.000
     3.000              gives  1 to b
                        holds for 3.000, until 6.000
     6.000 c 1          seizes 2 of b
                        holds for 2.000, until 8.000
           p 1          gives  1 to b
                        holds for 3.000, until 9.000
     8.000 c 1          seizes 2 of b
                        holds for 2.000, until 10.000
     9.000 p 1          gives  1 to b
                        holds for 3.000, until 12.000
    10.000 c 1          awaits 2 of b
    12.000 p 1          gives  1 to b
                        holds for 3.000, until 15.000
           c 1          seizes 2 of b
                        holds for 2.000, until 14.000
    14.000              awaits 2 of b
    15.000 p 1          gives  1 to b
                        holds for 3.000, until 18.000
    18.000              gives  1 to b
                        holds for 3.000, until 21.000
           c 1          seizes 2 of b
                        holds for 2.000, until 20.000
    20.000              awaits 2 of b
    21.000 p 1          gives  1 to b
                        holds for 3.000, until 24.000
    24.000              gives  1 to b
                        holds for 3.000, until 27.000
           c 1          seizes 2 of b
                        holds for 2.000, until 26.000
```

```
                         clock time =     25.000
**********************************************************************
*                                                                    *
*                             r e p o r t                            *
*                                                                    *
**********************************************************************


                                b i n s
                                *******

title       /   (re)set/   obs/init/ max/ now/ av. free/ av. wait/qmax
b                 0.000      8    2    3    0     1.120     2.000    1
```

```
                  -------------------- program 19 output --------------------


                              clock time =      0.000
**********************************************************************
*                                                                    *
*                t r a c i n g   c o m m e n c e s                   *
*                                                                    *
**********************************************************************

     time/ current     and its action(s)

    0.000 demos        schedules p 1 now
                       schedules p 2 at 5.000
                       schedules c 1 now
                       schedules c 2 at 1.000
                       schedules c 3 at 2.000
                       schedules c 4 at 3.000
                       holds for 35.000, until 35.000
          p 1          waits in waiting
          c 1          waits in waiting
          p 1          coopts c 1 from waiting
                       holds for 2.000, until 2.000
    1.000 c 2          waits in waiting
    2.000 c 3          waits in waiting
          p 1          schedules c 1 at 4.000
                       coopts c 2 from waiting
                       holds for 2.000, until 4.000
    3.000 c 4          waits in waiting
    4.000 c 1          waits in waiting
          p 1          schedules c 2 at 8.000
                       coopts c 3 from waiting
                       holds for 2.000, until 6.000
    5.000 p 2          coopts c 4 from waiting
                       holds for 2.000, until 7.000
    6.000 p 1          schedules c 3 at 12.000
                       coopts c 1 from waiting
                       holds for 2.000, until 8.000
    7.000 p 2          schedules c 4 at 14.000
                       waits in waiting
    8.000 c 2          waits in waiting
          p 1          schedules c 1 at 16.000
                       waits in waiting
          p 2          coopts c 2 from waiting
                       holds for 2.000, until 10.000
   10.000              schedules c 2 at 20.000
                       waits in waiting
   12.000 c 3          waits in waiting
          p 1          coopts c 3 from waiting
                       holds for 2.000, until 14.000
   14.000 c 4          waits in waiting
```

```
p 1          schedules c 3 at 28.000
             waits in waiting
p 2          coopts c 4 from waiting
             holds for 2.000, until 16.000
```

```
    16.000 c 1          waits in waiting
           p 2          schedules c 4 at 32.000
                        waits in waiting
           p 1          coopts c 1 from waiting
                        holds for 2.000, until 18.000
    18.000              schedules c 1 at 36.000
                        waits in waiting
    20.000 c 2          waits in waiting
           p 2          coopts c 2 from waiting
                        holds for 2.000, until 22.000
    22.000              schedules c 2 at 44.000
                        waits in waiting
    28.000 c 3          waits in waiting
           p 1          coopts c 3 from waiting
                        holds for 2.000, until 30.000
    30.000              schedules c 3 at 60.000
                        waits in waiting
    32.000 c 4          waits in waiting
           p 2          coopts c 4 from waiting
                        holds for 2.000, until 34.000
    34.000              schedules c 4 at 68.000
                        waits in waiting


                       clock time =     35.000
**********************************************************************
*                                                                    *
*                            r e p o r t                             *
*                                                                    *
**********************************************************************



                        w a i t   q u e u e s
                        ********************

title       /   (re)set/   obs/ qmax/ qnow/ q average/zeros/  av. wait
waiting           0.000     12     2     2      1.171     5      2.917
waiting     *     0.000     12     3     0      0.200     8      0.583
```

```
-------------------- program 20 output --------------------


                      clock time =    100.000
**********************************************************************
*                                                                    *
*                            r e p o r t                             *
*                                                                    *
**********************************************************************



                     d i s t r i b u t i o n s
                     *************************

title        /   (re)set/   obs/type     /        a/         b/     seed
cpu burst          0.000     88 negexp       1.000              33427485
next job           0.000     88 negexp       0.875              22276755


                           t a l l i e s
                           *************

title        /   (re)set/   obs/  average/est.st.dv/  minimum/  maximum
thru times         0.000     88     4.696      3.484 9.208e-02    11.267


                       w a i t   q u e u e s
                       *********************

title        /   (re)set/   obs/ qmax/ qnow/ q average/zeros/  av. wait
cpu q              0.000     88     1     1  9.892e-02    76      0.111
cpu q        *     0.000     88    10     0      3.231    13      3.672
```

appendix i

i n d e x