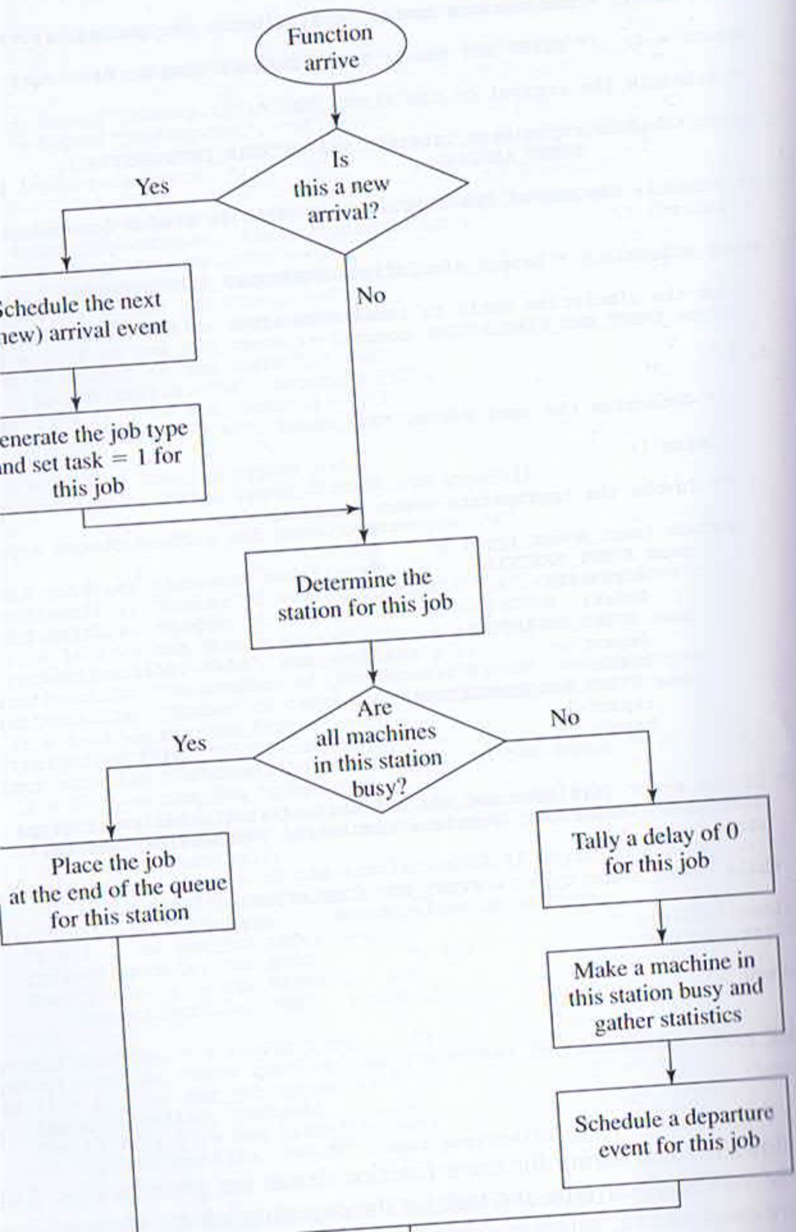


```
void arrive(int new_job)  /* Function to serve as both an arrival event of a job
                             to the system, as well as the non-event of a job's
                             arriving to a subsequent station along its
                             route. */
{
    int station;

    /* If this is a new arrival to the system, generate the time of the next
       arrival and determine the job type and task number of the arriving
       job. */

    if (new_job == 1) {

        event_schedule(sim_time + expon(mean_interarrival, STREAM_INTERARRIVAL),
                       EVENT_ARRIVAL);
        job_type = random_integer(prob_distrib_job_type, STREAM_JOB_TYPE);
        task     = 1;
    }

    /* Determine the station from the route matrix. */

    station = route[job_type][task];

    /* Check to see whether all machines in this station are busy. */

    if (num_machines_busy[station] == num_machines[station]) {

        /* All machines in this station are busy, so place the arriving job at
           the end of the appropriate queue. Note that the following data are
           stored in the record for each job:
             1. Time of arrival to this station.
             2. Job type.
             3. Current task number. */

        transfer[1] = sim_time;
        transfer[2] = job_type;
        transfer[3] = task;
        list_file(LAST, station);
    }

    else {

        /* A machine in this station is idle, so start service on the arriving
           job (which has a delay of zero). */

        sampst(0.0, station);                               /* For station. */
        sampst(0.0, num_stations + job_type);               /* For job type. */
        ++num_machines_busy[station];
        timest((float) num_machines_busy[station], station);

        /* Schedule a service completion.  Note defining attributes beyond the
           first two for the event record before invoking event_schedule. */

        transfer[3] = job_type;
        transfer[4] = task;
        event_schedule(sim_time
                       + erlang(2, mean_service[job_type][task],
                                STREAM_SERVICE),
                       EVENT_DEPARTURE);
```

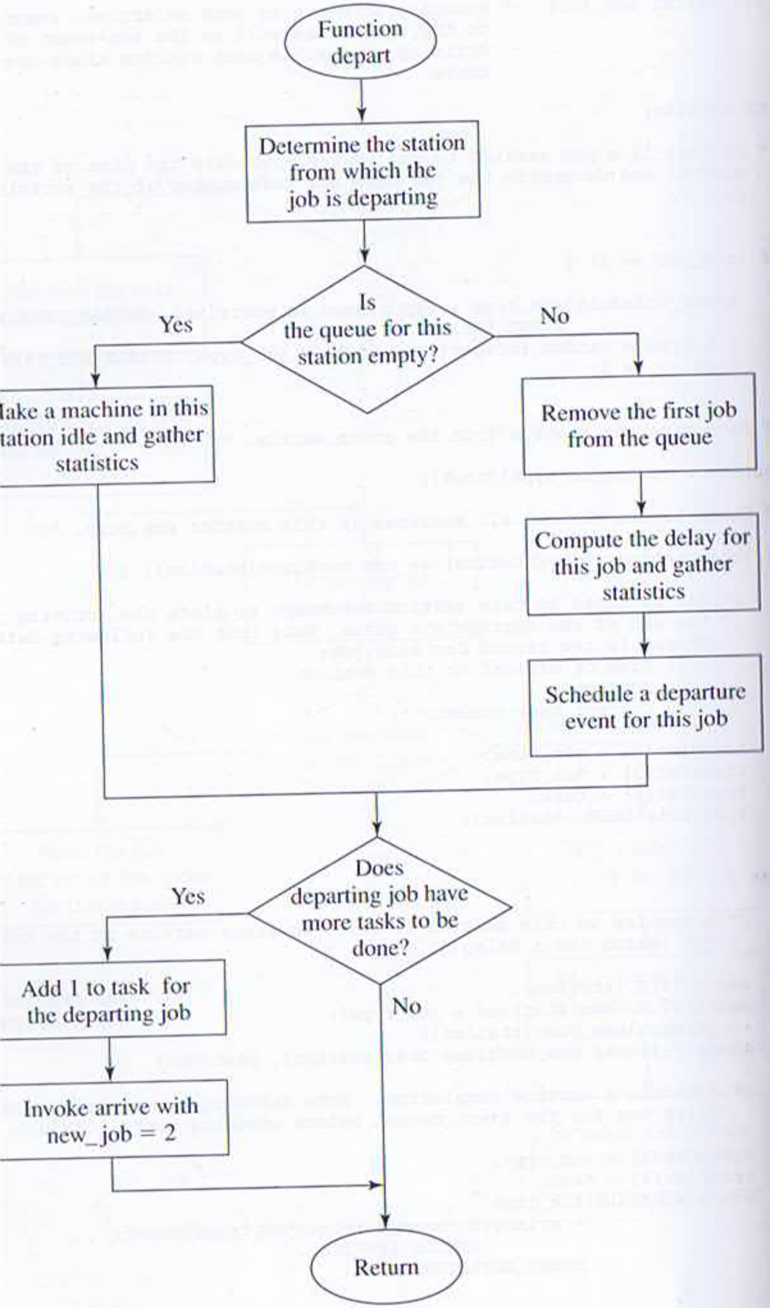

FIGURE 2.43
Flowchart for departure function, job-shop model.

```
void depart(void)  /* Event function for departure of a job from a particular
                      station. */
{
    int station, job_type_queue, task_queue;

    /* Determine the station from which the job is departing. */

    job_type = transfer[3];
    task     = transfer[4];
    station  = route[job_type][task];

    /* Check to see whether the queue for this station is empty. */

    if (list_size[station] == 0) {

        /* The queue for this station is empty, so make a machine in this
           station idle. */

        --num_machines_busy[station];
        timest((float) num_machines_busy[station], station);
    }

    else {

        /* The queue is nonempty, so start service on first job in queue. */

        list_remove(FIRST, station);

        /* Tally this delay for this station. */

        sampst(sim_time - transfer[1], station);

        /* Tally this same delay for this job type. */

        job_type_queue = transfer[2];
        task_queue     = transfer[3];
        sampst(sim_time - transfer[1], num_stations + job_type_queue);

        /* Schedule end of service for this job at this station.  Note defining
           attributes beyond the first two for the event record before invoking
           event_schedule. */

        transfer[3] = job_type_queue;
        transfer[4] = task_queue;
        event_schedule(sim_time
                       + erlang(2, mean_service[job_type_queue][task_queue],
                                STREAM_SERVICE),
                       EVENT_DEPARTURE);
    }

    /* If the current departing job has one or more tasks yet to be done, send
       the job to the next station on its route. */

    if (task < num_tasks[job_type]) {
        ++task;
        arrive(2);
    }
}
```

FIGURE 2.44
Code for the function depart, job-shop model.

```
ort(void)  /* Report generator function. */

    i;
at overall_avg_job_tot_delay, avg_job_tot_delay, sum_probs;

Compute the average total delay in queue for each job type and the
overall average job total delay. */

intf(outfile, "\n\n\n\nJob type      Average total delay in queue");
rall_avg_job_tot_delay = 0.0;
_probs                 = 0.0;
 (i = 1; i <= num_job_types; ++i) {
  avg_job_tot_delay = sampst(0.0, -(num_stations + i)) * num_tasks[i];
  fprintf(outfile, "\n\n%4d%27.3f", i, avg_job_tot_delay);
  overall_avg_job_tot_delay += (prob_distrib_job_type[i] - sum_probs)
                               * avg_job_tot_delay;
  sum_probs = prob_distrib_job_type[i];

intf(outfile, "\n\nOverall average job total delay =%10.3f\n",
     overall_avg_job_tot_delay);

 Compute the average number in queue, the average utilization, and the
 average delay in queue for each station. */

rintf(outfile,
      "\n\n\n Work      Average number      Average       Average delay");
rintf(outfile,
        "\nstation       in queue         utilization      in queue");
r (j = 1; j <= num_stations; ++j)
  fprintf(outfile, "\n\n%4d%17.3f%17.3f%17.3f", j, filest(j),
          timest(0.0, -j) / num_machines[j], sampst(0.0, -j));
```

RE 2.45
for the function report, job-shop model.

ouble to avoid excessive roundoff error in the subtraction for the delay calcu-. Finally, if the job leaving this station still has more tasks to be done, its task er is incremented and it is sent on its way to the next station on its route by ing arrive, now with new_job set to 2 to indicate that this is not a newly arriv- b.

he code for the report-generator function is in Fig. 2.45. The first for loop utes the average total delay in all the queues for each job type i; the word " is used here to indicate that this is to be the average delay summed for *all* the es along the route for each job type. We must multiply the average returned mpst by the number of tasks for this job type, num_tasks[i], since sampst nvoked for each job of this type that left the system num_tasks[i] times rather once, so that the denominator used by sampst to compute the average is _tasks[i] times too large. We then weight these average total delays by the prob- ies for the job types and add them up to get the overall average job total delay; se these true (exact) probabilities of job types to obtain a more precise (less ble) estimate than if we simply averaged all the job total delays regardless of ype. Also, we must take successive differences in the prob_distrib_ job_type

the sampst average by num_tasks[i] is slightly incorrect. Since there will generally be some jobs left in the system at the end of the simulation that have not experienced their delays in all the queues, they should not have had *any* of their delays registered in sampst. However, since this simulation is $365 \times 8 = 2920$ hours long and since there are 4 job arrivals expected each hour, there will be an expected 11,680 job arrivals, so this error is likely to be minor. See Prob. 2.5 for an alternative way of collecting the total delays in queue by job type, which avoids this difficulty.) The function closes with a for loop to write out, for each station j, the time-average number in queue, the utilization (computed as the time-average number of machines busy divided by the number of machines in the station), and the average delay in queue.

The function to generate an m-Erlang random variable is in Fig. 2.66 in App. 2A (it is part of simlib, not this model), and follows the physical model for the Erlang distribution described earlier. Note that we must divide the desired expectation of the final Erlang random variable by m to determine the expectation of the component exponential random variables. Also, the user-specified stream number, "stream", is taken as input here and simply passed through to the exponential generator (Fig. 2.63 in App. 2A) and then on to the random-number generator lcgrand.

2.7.3 Simulation Output and Discussion

Figure 2.46 shows the output file (jobshop.out) for this simulation. Weighted by job type, the average time spent by jobs waiting in the queues was almost 11 hours; this

```
Job-shop model

Number of workstations                  5

Number of machines in each station      3     2     4     3     1

Number of job types                     3

Number of tasks for each job type       4     3     5

Distribution function of job types   0.300   0.800   1.000

Mean interarrival time of jobs         0.25 hours

Length of the simulation             365.0 eight-hour days


Job type     Workstations on route

    1           3     1     2     5

    2           4     1     3

    3           2     5     1     4     3
```

FIGURE 2.46