In this lab you will simulate scheduling in order to see how the time required depends on the scheduling algorithm and the request patterns. The lab is due in 2 weeks, 14 October, 2009.

A process is characterized by just four numbers A, B, C, and IO. Please do not confuse IO with the number 10. A is the arrival time of the process and C is the total CPU time needed. A process contains computation alternating with I/O. We refer to these as CPU bursts and I/O bursts. We make the simplifying assumption that, for each process, the CPU burst times are uniformly distributed random integers (UDRIs) in the interval (0,B]. To obtain a URDI t in some interval (0,U] use the function randomOS(U) described below. If the t chosen is larger that the total CPU time remaining, reduce t to the remaining time. Similarly, we assume the I/O times are UDRIs in the interval (0,IO].

You are to read a file describing n processes (i.e., n quadruples of numbers) and then simulate the n processes until they all terminate. The way to do this is to keep track of the state of each process and advance time making any state transitions needed. At the end of the run you should first print an identification of the run including the scheduling algorithm used, any parameters (e.g. the quantum for RR), and the number of processes simulated. You should then print for each process

- (A, B, C, IO)
- Finishing time.
- Turnaround time (finishing time - A).
- I/O time (i.e., time in Blocked state).
- Waiting time (i.e., time in Ready state).

Then print the following summary data.

- Finishing time (i.e., when all the processes have finished).
- CPU Utilization (i.e., percentage of time some job is running).
- I/O Utilization (i.e., percentage of time some job is blocked).
- Throughput, expressed in processes completed per hundred time units.
- Average turnaround time.
- Average waiting time.

You should simulate each of the following scheduling algorithms, assuming, for simplicity, that a context switch takes zero time. You need only do calculations every time unit (e.g., you may assume nothing exciting happens at time 2.5).

- FCFS.
- RR with quantum 2.
- PSJF. Recall that PSJF is shortest process next, not shortest burst next. So the time you use to determine priority is the total time remaining (i.e., the input value C minus the number of cycles this process has run).
- HPRN. Define the denominator to be max(1, running time), to prevent dividing by zero for a job that has yet to be run. Remember that HPRN is non-preemptive.

For each scheduling algorithm there are several runs with different process mixes. A mix is a value of n followed by n (A, B, C, IO) quadruples. Here are the first two input sets. The comments are *not* part of the input. All the input sets, with the corresponding outputs are on the web.

```
1 (0 1 5 1)  about as easy as possible
2 (0 1 5 1) (0 1 5 1)  should alternate with FCFS
```

The function randomOS(U) reads a random non-negative integer X from a file named random-numbers (in the current directory) and returns the value 1+(X mod U). I will supply a file with a large number of random non-negative integers. The purpose of standardizing the random numbers is so that your program will produce the published answers.

**Breaking ties:**
There are three places where the above specification is not deterministic and different choices can lead to different answers. To standardize the answers we make the following choices.

1. A running process can have three events occur.
   i. It can terminate (CPU time remaining goes to zero).
   ii. It can block (the current CPU burst goes to zero).
   iii. It can be preempted (e.g., the RR quantum goes to zero).

They should be processed in the above order. For example if all three occur at one cycle, the process terminates.

2. Many jobs can have the same "priority". For example in RR jobs can become ready at the same cycle and you must decide in what order to insert them onto the FIFO ready queue. Ties should be broken by favoring the process with the earliest arrival time. If the arrival times are the same for two processes with the same priority, then favor the process that is listed earliest in the input.

3. A random number is chosen at two points.
   i. When a running process is blocked (to determine the I/O-burst).
   ii. When a ready process is run (to determine the cpu-burst).

If both events occur during the same cycle, process them in the above order.

**Submitting the lab:**
Please read and follow the guidelines on the web page.

**Running your program**
Your program must read its input from a file, whose name is given as a command line argument. The format of the input is shown above (but the "comments" are not necessary); sample inputs are on the web. Be sure you can process the sample input. Do not assume you can change the input by adding spaces or commas or removing parens, etc.

In addition your program must accept an optional "--verbose" flag, which if present precedes the file name. If --verbose is given your program is to produce detailed output that you will find useful in debugging (indeed you will thank me for requiring this). See the sample output on the web for the format of debugging output. We may find the debugging data useful if your program has errors. So the two possible invocations of your program are

<program-name> <input-filename>
<program-name> --verbose <input-filename>

My program also supports an even more verbose mode (show-random) that prints the random number chosen each time. This is useful but your program is **not** required to support it.