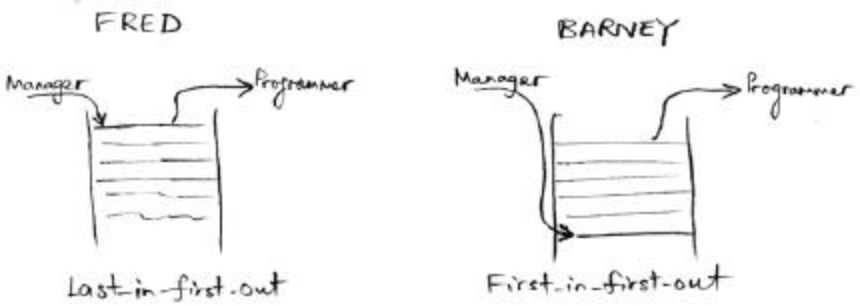
Sample question 1:

Consider a work situation that involves two managers called Fred and Barney and two programmers. The first programmer receives instructions from Fred whereas the second programmer receives instructions from Barney. These two teams work on separate projects. There are two separate 'in-trays' from which the two programmers receive their respective instructions. When each programmer finishes a task, he takes the next job from the top of his in-tray. Fred's practice is to place the next work order on top of his programmer's in-tray whereas Barney always places the next task at the bottom of his programmer's in-tray. (see the diagrams below).



Write a computer simulation program either in C, C++ or Java to obtain an estimate of the average and standard deviation of the total delivery time per task for each of the processing sequences described above. The total delivery time refers to the sum of the waiting time and the programming time. The 'in tray' must be represented as a linked list for both cases.

For the purpose of this exercise, assume that the inter-task arrival time is evenly distributed between 4 and 7 days in real time for both cases. Similarly, assume that the time required by each programmer to accomplish each programming task is evenly distributed between 3.9 and 6.9 days in real time. Obtain the required estimates based on 200,000 simulated tasks.

For those who did not study Statistics, the standard deviation of the total delivery time is given by the following expression:

$$S = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n} - \overline{X}^2}$$

where x_i, $\overline{X}$ and n denote respectively the total delivery time for the ith task, average total delivery time per task and number of tasks.

Sample question 2:

At some time in the future a research company called Gammaswitch has invented a new ultra high-speed computing device that operates by switching high-energy photons instead of switching electrical charge with semiconductor devices. The Gammaswitch computer is over a thousand times faster than the fastest semiconductor devices.

An entrepreneurial young hacker by the name of Bob Yates has created the only commercially viable operating system for the Gammaswitch computer. It is an architecturally challenged, barely functional monster called Braindead.

An Internet information service called Regurgital operates a worldwide free-to-wire information service. Regurgital's service has become so popular that its message server has become overloaded and is causing severe delays and inconvenience to Regurgital's customers.

Regurgital now wants to replace the overloaded computer with one of Gammaswitch's new ultra-high speed machines. Your job is to program the Gammaswitch computer in C or C++, but in any case, under the frightful Braindead operating system, to copy the information messages from disk files on the Gammaswitch computer and deliver them to Braindead's HTTP driver.

Your program must be able to handle up to 20 concurrent HTTP downloads.

If you write in C or C++, Braindead provides the following HTTP server support functions.

GetRequest	initiates reception of an HTTP request.
PutData	initiates transmission of a block of output text.
RequestDone	tells Braindead that the request has been completely serviced.
Yield	releases control to Braindead.

GetRequest has the following declaration.

```
int GetRequest ();
```

GetRequest returns an integer known as a 'channel identifier'. The channel identifier is used to relate subsequent network operations to the invitation to receive an HTTP connection created by the GetRequest call. Multiple calls to GetRequest may be made one after another. Each open channel will be assigned a different channel number by GetRequest.

The GetRequest function merely establishes an invitation to receive an HTTP connection. It does not actually create a connection to a client computer. When, in due course, a client computer attempts to connect to the HTTP server, Braindead will call an application function called 'RequestReady' to notify the application that a connection has been made. The application must declare a function called RequestReady or it will fail to link. RequestReady must have the following declaration.

```
void RequestReady (int chanId,
          const char *fileName);
```

The argument 'chanId' will contain the channel identifier returned by the call to GetRequest that created the communications channel.

The argument 'fileName' is a pointer to the name of the file to be returned to the client computer. The memory addressed by fileName will only contain the file name during the execution of RequestReady. When RequestReady returns, Braindead may overwrite the text of the file name.

The application can use the normal C fopen, fread and fclose functions to read the file from the mass storage attached to the Gammaswitch computer.

To return the file contents to the requesting client, the application must call Braindead's PutData function. PutData has the following declaration.

```
void PutData (int chanId
          const char *data, int len);
```

The argument 'chanId' is the channel identifier returned by the original call to GetRequest.

The argument 'data' is a pointer to a data block that contains the data to be written.

The argument 'len' is the length of the data block.

The call to PutData merely initiates a data transfer. PutData will usually return before transmission of the data is finished. When transmission of the data is completed, Braindead calls an application function called 'PutDone' to notify the application that the transmission process is completed.

The declaration of PutDone is:

```
void PutDone (int chanId);
```

The argument 'chanId' is the channel identifier returned by the original call to GetRequest.

Braindead does not copy the contents of the data block passed to PutData into an internal buffer. Instead it stores a pointer to the data block in the application's memory segment. Consequently, it is important that the application does not modify the data block or release the data block memory by returning from a function in which it is declared as a local variable before Braindead calls the PutDone function.

When the file has been completely sent by repeated calls to PutData, the application must call the RequestDone function to inform Braindead that the request has been completely serviced. RequestDone has the following declaration.

```
void RequestDone (int chanId);
```

The argument ‘chanId’ is the channel identifier originally returned by GetRequest.

RequestDone closes the communications channel created by GetRequest and releases the channel identifier. To receive another request, the application must make a further call to GetRequest. Once RequestDone has released a channel identifier, it is possible that a subsequent call to GetRequest may return that same channel identifier.

In order to give Braindead the opportunity to perform its own internal processing and call the GetReady and PutDone functions, the application must release control to Braindead by calling the Yield function. The declaration of Yield is:

```
void Yield ();
```

When the application calls Yield, Braindead scans the devices that perform the physical input and output and checks whether an input or output process has been completed. If such a process has finished, Braindead calls either GetReady or PutDone depending on the nature of the original request. If an input or output process is outstanding, Yield will return after Braindead has called either GetReady or PutDone. If no input or output processes are outstanding, Yield will return immediately.

The declarations of the Braindead functions and application call back entry points are contained in the file ‘braindead.h’. A simulator for Braindead under Unix or Windows is contained in the file ‘braindead.c’.