Here are the pre-defined data structures and procedures for the Scheduler pseudocode.

RunningProcess - global variable which contains the PCB for the current running process.

IO_DonePID - global variable which contains the process id PID of the process whose I/O just completed.

PCB - a structure for the process control block. It contains the following fields:
  - PID - process ID
  - Next_PCB - link to the next PCB in the queue
  - PC - current program counter value for this process
  - Other - you can define other fields if you need them

ReadyQueue - linked list of PCBs of processes waiting for the CPU.

WaitQueue - linked list of PCBs of processes waiting for I/O to complete.

Append(pcb, queue) - appends pcb to tail of queue.

Head(queue) - removes the first PCB in queue and returns it; if queue is empty it returns NIL.

Select(queue,pid) - searches queue for the first process PCB whose PID matches pid.

ContextSwitch(pcb,TAU) - set the interval timer clock to TAU and start executing the process whose PCB is specified.
NOTE: ContextSwitch never returns. The PC (Program Counter) is loaded with the PC value stored in pcb->PC which causes the fetch/execute cycle to start where that process left off. This is sufficient to get the new process running.

ReadTime() - returns the time of day (e.g. 22:10:03 in hours:minutes:seconds)

idle() - if the ReadyQueue is empty, your pseudo-code can call idle() and the CPU will idle until some kind of interrupt occurs.

You can define any other procedures or data structures you need, but these should be sufficient.
Scheduler Pseudo-code Template:

scheduler() {
    if (called by a process that is finished) {
        /* current process is done */
    } else {
        if (called by I/O routines) {
            /* current process requests I/O */
        } else {
            if (called by the I/O interrupt handler) {
                /* I/O done for process IO_DonePID */
            } else {
                if (called by the timer interrupt handler) {
                    /* current process has used up it's time quantum */
                } else {
                    /* current process has used up it's time quantum */
                }
            }
        }
    }
}