1 Pseudocode

```
Initialize the queues (Job Dispatch and Level 0-2 queues);
Fill Job Dispatch queue from job dispatch list file;
Ask the user to enter integer values for parameters;
While there is a currently running process or either queue is not empty
    Unload any arrived pending processes from the Job Dispatch queue;
    Dequeue process from Job Dispatch queue and Enqueue on Level 0 queue;
    If a process is currently running
        Decrease process remaining_cpu_time by quantum;
        If times up
            Send SIGINT to the process to terminate it;
            Calculate the turnaround time;
            Update the overall turnaround time;
            Free up process structure memory;
        Else
            Send SIGTSTP to suspend currently running process;
            If the process is in Level 0 queue
                Enqueue it to the tail of Level 1 queue;
            Else if it is in Level 1 queue
                Increase the iteration counter by 1;
                If the counter is over the iteration limit
                    Enqueue it to the tail of Level 2 queue;
                Else
                    Enqueue it back to the tail of Level 1 queue again;
            Else if it is in Level 2 queue
                Enqueue it back to the head of Level 2 queue;
        Mark the process has been run;
    If no process is currently running
        Dequeue a process from the head of the first available non empty queue;
        Mark the corresponding level the process is at;
        If the process job is a suspended process
            Send SIGCONT to resume it;
        Else
            Start it (fork & exec);
    Set the process as the currently running process;
    If there's a process to run
        Update the quantum by the smaller one between the cpu remaining time \
        and the corresponding time quantum per the queue the process is at;
    Else
        Set the quantum as the minimum value among t_0, t_1 and 1;
    Sleep for quantum;
    Increase timer by quantum;
    Go back to the while loop;
Calculate and show the average turnaround time and the average waiting time;
Terminate the Round Robin dispatcher;
```

2 Testing Boundary Value Analysis

We use the following notations:

- $\hat{t_0}$: time quantum for the level 0 queue
- $\hat{t_1}$: time quantum for the level 1 queue
- k: RR iteration upper bound for the level 1 queue
- A, C: arrival time and overall CPU time for task i
- t_0, t_1, t_2: CPU time spent to execute a fraction of a task as a level 0-2 process.

As a direct result, $C = \sum_{i=1,2,3} t_i$. Now let's build the test bundle in a bottom up manner.

2.1 Quantum calculation test cases

These are unit tests for checking the correctness of dynamic values of 'quantum' variable calculation.

1. given a task with $C < \hat{t_0}$
2. given a task with $(C - \hat{t_0}) \bmod k \not\equiv 0$ and $\hat{t_0} < C < \hat{t_0} + k\hat{t_1}$
3. given a task with $C > \hat{t_0} + k\hat{t_1}$

2.2 Simple chronological test cases

These are for testing if pre-empts work and processes are put into a queue with a correct priority. (Assume that tasks $\leq i - 1$ have been complete and $> i + 1$ won't arrive before the completion of $i + 1$.)

1. $A_{i+1} < A_i + t_{i0}$
2. $A_i + t_{i0} < A_{i+1} < A_i + t_{i0} + t_{i1}$
3. $A_{i+1} > A_i + t_{i0} + t_{i1}$
4. $A_i + t_{i0} < A_{i+1} + t_{(i+1)0} < A_i + t_{i0} + t_{i1}$
5. $A_{i+1} + t_{(i+1)0} > A_i + t_{i0} + t_{i1}$
6. $A_i + t_{i0} + t_{i1} < A_{i+1} + t_{(i+1)0} + t_{(i+1)1} < A_i + t_{i0} + t_{i1} + t_{i2}$
7. $A_i + t_{i0} + t_{i1} + t_{i2} >, =, < A_{i+1} + t_{(i+1)0} + t_{(i+1)1} + t_{(i+1)2}$

2.3 Miscellaneous test cases

- when $k = 1, k > 2$ if the iteration capacity works correctly.(Enqueue back in level 1 or to level 2).
- test if the tasks still on level 2 queue are put in the head of the level 2 queue instead of the tail.
- Some short duration tasks happen to arrive when the other process just run in a quantum.(waiting time = 0).
- The processor is idle for a while between the completion of task i and the arrival of task $i + 1$.
- Mutiple tasks arrive at the same time
- Frequent arrivals of tasks with long cpu time.

Submitted test files under 'tests' folder cover all cases mentioned in section 2.1-2.3. But you need to mannually try different $[\hat{t_0}, \hat{t_1}, k]$ parameters to test all combinations of the partitions designed. For 'test_file_1', I used $[3, 2, 1]$, $[3, 1, 2]$, $[2, 3, 3]$. For 'test_file_2', $[2, 1, 1]$, $[1, 1, 2]$, $[1, 2, 3]$. For 'test_file_3':, $[3, 2, 5]$, $[3, 2, 1]$, $[2, 3, 2]$