1 Pseudocode

1.1 The main program

```
Initialize the queues (job queue, memory management queue and Level 0-2 queues);
Fill Job Dispatch queue from job dispatch list file;
Ask the user to enter integer values for parameters;
Initialize the root memory block of size 2048(main memory to use);
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 arrived job queue;
    If there's a task on the arrived job queue
        dequeue the task;
        try to allocate the task on the main memory to a block fitting its size;
        if a memory block is assigned to the task
            store the pointer of the block in a PCB field;
            enqueue the process in level 0 queue;
        else there's no block available
            queue the task back in the front of the arrived job queue again;
    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 the memory block the process was allocated to;
            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;
```

1.2 MAB subroutines

- allocate a process to a memory block of an appropriate size

```
If the size of the process is less than 8 (MB)
    Pad the size to be 8;
Call the block splitting oracle routine
to find a suitable block on the main memory;
If there's a free block
    Mark the block as being in use;
    Return the pointer of the block;
Else
    Return a null pointer
```

- block split oracle

```
//Base Case:
If the block is in use
    Return a null pointer;
If the block can't be split evenly into 2 blocks to store the process
    If a part of the block is not free
        Return a null pointer;
    Else
        Return the pointer of the block;
//Recursive Step:
If the block has been split already
    Attempt to search down from its left part;
    If there's no space under its left child block
        Attempt to search down from its right part;
Else
    Create two children blocks evenly;
    Attempt to use this oracle to split its left child further;
```

- free the memory block a process occupies

```
Mark the block as being free;
Call the merge oracle routine to handle the merging protocol.
```

- merge oracle

```
//Base Case:
If the block is the root
    Do nothing and return;
//Recursive Step:
If the sibling block is free
(implicitly there's no children for the sibling block)
    Delete the block and its sibling block;
    Merge 2 blocks into 1;
    Use the oracle itself to attempt to merge the combined block;
```

2 Testing Boundary Value Analysis

We use the notation D_i for the memory size requirement for a task i.
Chronological testing has been conducted previously in Stage 1, therefore, processes' time sequences are kept the same. A third field standing for size is appended. Here focused are memory allocation and testing the strictly FCFS blocking mode of the arrived job queue.

2.1 Buddy memory allocation API unit testing

Here we use MAB_MAX (the overall size of the main memory) as 2048 (MB), MAB_MIN (lower bound size of a block) as 8 (MB).

2.1.1 Alloc

1. allocate a block of size $7 > D_i \geq 1; \forall j \in [3, ..., 10] : 2^{j+1} > D_i \geq 2^j; D_i = 2048$ to an empty main memory
2. allocate $0 < D_i \leq 2048$ when there's a block of 1025 MB in use.
3. allocate $D_i = 1024; D_i < 1024; D_i > 1024$ when there's a block of ≤ 1024 MB in use.
4. allocate 16 blocks of 128 MB to test multiple even blocks
5. allocate 16 blocks of 129 MB to test slightly screwed blocks.

2.1.2 Free

1. free the main memory in use/free
2. free any block not in use(only testable calling a dummy Mab object directly).
3. free the only block in use(free the pointers returned upon success allocation in section 2.1.1)

2.1.3 Alloc & Free combination to test Spilt and Merge in a normal workflow

Please check the online instance. You can find a testing C file('test_mab.c') and replicate the unit testing by 'gcc -o <unittest> mab.c pcb.c test_mab.c' and run the executable '<unittest>'.

2.2 Arrived job queue testing

1. a process of 1025 MB to block the (chronologically) following processes before that process is terminated
2. 2 consecutive processes of 1024 MB to run concurrently.
3. multiple small processes that are not blocked.

Note that user inputs tuning matters to test. For example, 'test_file_2' I recommend to use a large number for the iteration cap and 1 for the level 0 quantum to monitor concurrent RRs for multiple processes with compatible size.