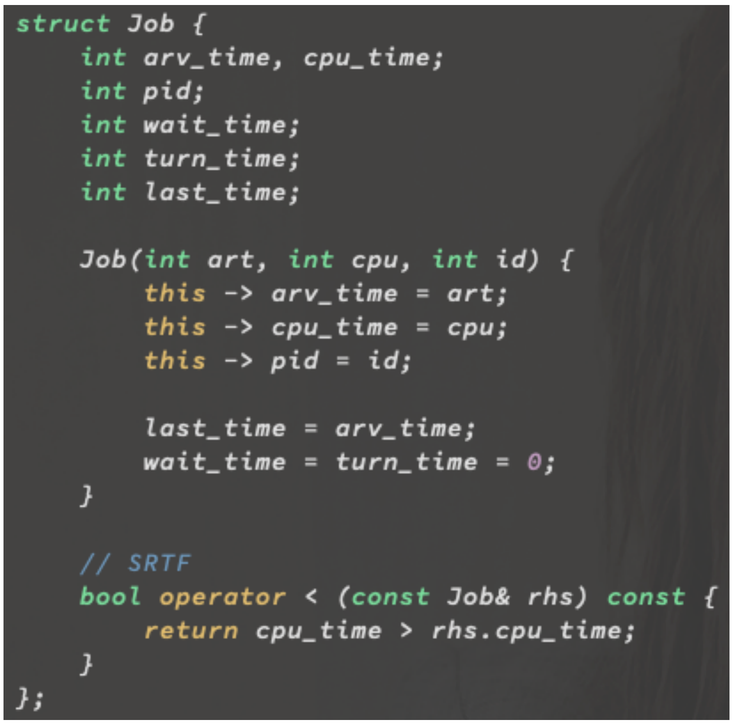
HW2 Report

Q1. Data structure

Variable to Information Table

Variable Name	Information
arv_name	Arrival Time
cpu_time	CPU Burst (Remaining Time)
pid	Process ID
wait_time	Waiting Time
turn_time	Turnaround Time
last_time	Time of start waiting

Table 1

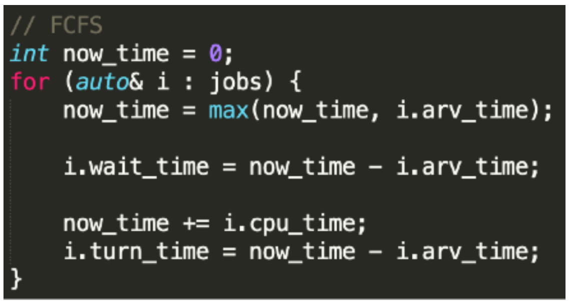


```
struct Job {
    int arv_time, cpu_time;
    int pid;
    int wait_time;
    int turn_time;
    int last_time;

    Job(int art, int cpu, int id) {
        this -> arv_time = art;
        this -> cpu_time = cpu;
        this -> pid = id;

        last_time = arv_time;
        wait_time = turn_time = 0;
    }

    // SRTF
    bool operator < (const Job& rhs) const {
        return cpu_time > rhs.cpu_time;
    }
};
```

I use structure to save every processes' informations (refer to Table 1). Each correspond to specific info. For *cpu_time* , it stands for CPU Burst at the beginning of each scheduling algorithm. And use *cpu_time* as remaining CPU burst for cases 3 (Round-Robin) and 4 (Round-Robin + SRTF: MultilevelFeedbackQueue).

Q2. Implementation

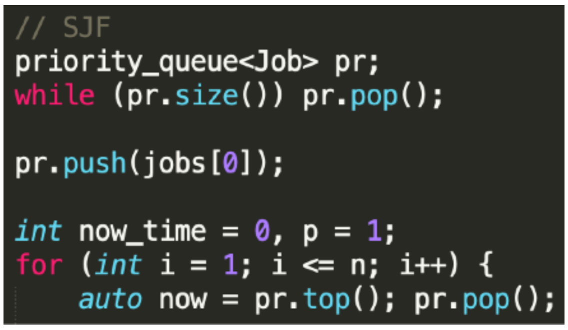
When the process scheduling algorithm has priority part, I would define the Job operator to follow the algorithm rule, using STL-priority queue to choose the one with highest priority. As for the FCFS or Round-Robin, STL-queue is the one that I used to simulate CPU queue.



```
// FCFS
int now_time = 0;
for (auto& i : jobs) {
    now_time = max(now_time, i.arv_time);

    i.wait_time = now_time - i.arv_time;

    now_time += i.cpu_time;
    i.turn_time = now_time - i.arv_time;
}
```

Code Segment of FCFS

Another problem is how to simulate the CPU work and calculate the informations we want to know. Take hw2_1 First-Come-First-Serve (FCFS) for example, use variable *now_time* as CPU time. Firstly, the jobs would store in vector *jobs* in the order of their arrival time. Because of the scheduling would let CPU serve accordingly, we just sum up the time and store back.



```
// SJF
priority_queue<Job> pr;
while (pr.size()) pr.pop();

pr.push(jobs[0]);

int now_time = 0, p = 1;
for (int i = 1; i <= n; i++) {
    auto now = pr.top(); pr.pop();
```

Code Segment of SJF

SJF (Shortest-Job-First) would choose the shortest CPU burst. Using priority queue and self define < operator to make job with shorter *cpu_time* with higher priority.

The other scheduling are all follow the above method to simulate the jobs selection. Combination of STL (queue and priority_queue) are very helpful in this simulation.

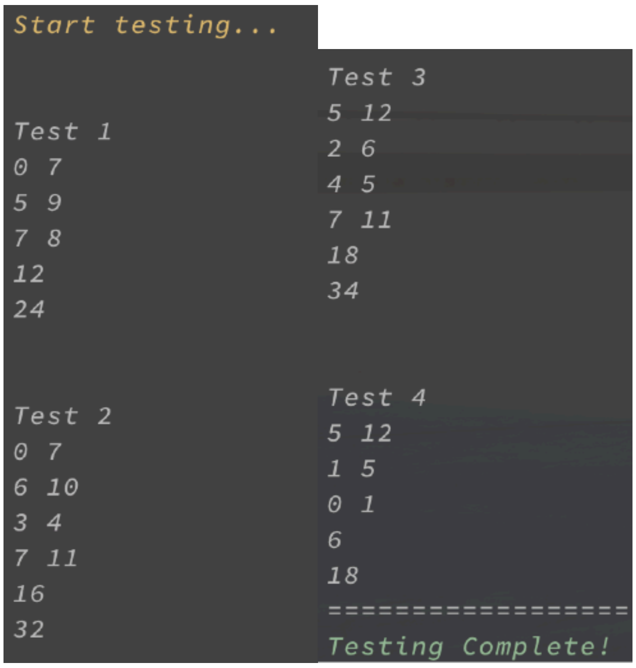
Q3. Problems & Solution

During the time working in this homework, the main problems is not knowing the code I wrote are totally correct or not. There is only one test case for each scheduling. The only solution is simulating the scheduling on my own and try to generate more test cases to ensure the code correctness. I think the TAs could make an OJ platform or provide more test cases.

Q4. Results & Conclusion

The test result for each scheduling algorithm are show at the end of this page. They are all match the outputs.

After this homework, I more clear about how this scheduling work and the calculation of different analysis information including waiting time and turn around time. Also, leaving deep impression on how this scheduling work in my brain.



```
Start testing...


Test 1
0 7
5 9
7 8
12
24


Test 2
0 7
6 10
3 4
7 11
16
32
```

```
Test 3
5 12
2 6
4 5
7 11
18
34


Test 4
5 12
1 5
0 1
6
18
==================
Testing Complete!
```