OS Term Project

-CPU Scheduling Algorithms Simulator-

COSE341(02)

7조

2017170327	건축사회환경공학부	강성묵
2019130456	심리학부	남정화
2020320010	컴퓨터학과	전병우

Intro: 계획 및 수행 역할

I. 팀 GitHub

https://github.com/deep-palette/CPU-scheduling-simulator

깃허브의 file update, commit 등을 활용하여 협업을 진행했다.

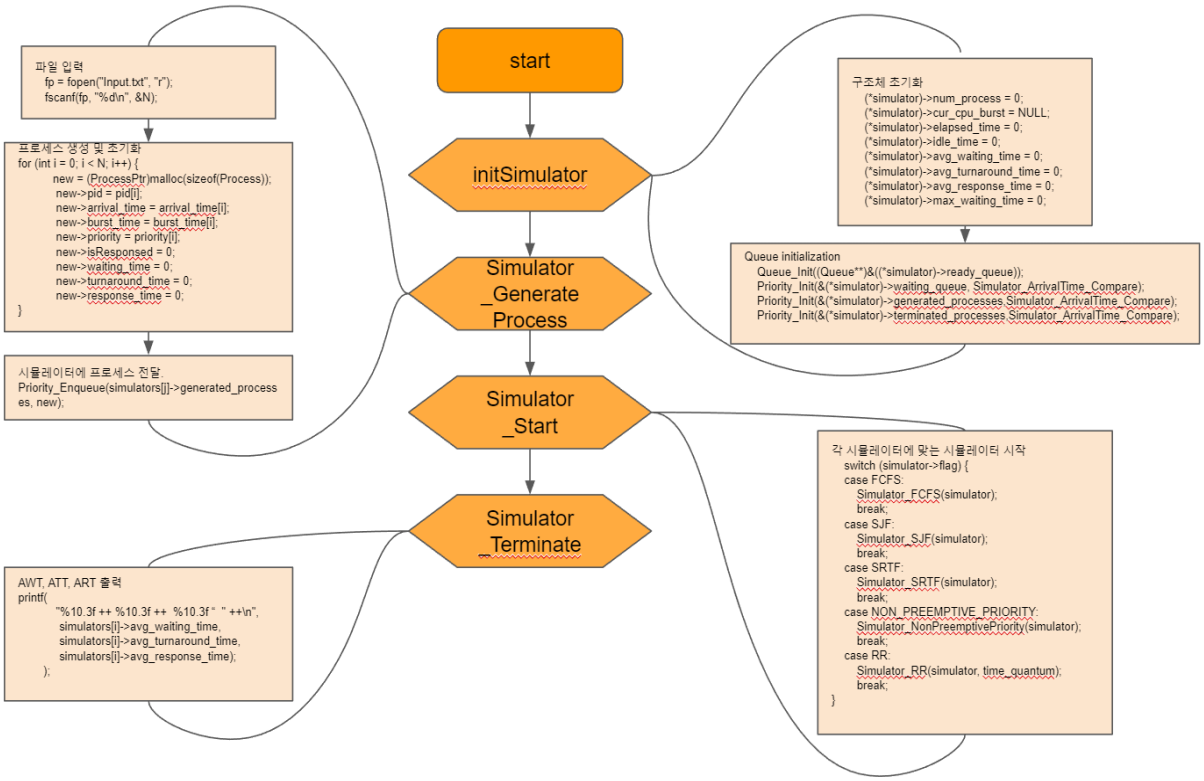
II. 팀원 역할

- 강성묵:자료구조, priority queue.c, queue.c 구현, 보고서 작성
- 남정화:발표, PPT제작, simulator.c 일부 구현(response time 로직, SRTF 오류 수정), 보고서 작성
- 전병우:algorithm, simulator.c, main.c 등 과제의 핵심 코드 구현, 보고서 작성

III. 개발 언어

C언어

IV. 함수 구성도



Simulator 자료구조 내 변수들을 초기화 한 다음, input file에서 process 정보를 모아 Process 구조체를 완성한다. 다음으로, CPU 스케줄링 알고리즘 5개를 실행해서 결과 정보를 얻고 이를 CLI에 출력하며 프로그램을 마치는 형태로 계획했다.

initSimulator - > Simulator_GenerateProcess -> Simulator_Start - > (Simulator_Evaluation) - > Simulator_Terminate (Print Simulator_Evaluation)

Body: 자료구조 및 주요변수 설명

-pseudo codes 및 동작-

I. Queues

Cpu에서 프로세스 작업을 처리 함에 있어 한번에 여러개의 프로세스 작업을 수행할 수 없다. 한번에 한개의 프로세스에서 작업을 수행 해야 하는데, 수행하고자 하는 프로세스가 여러개가 있다면 그 프로세스를 한 공간에 모아 대기 시키고 그 대기 공간에서 한번에 하나의 프로세스를 빼와 CPU에서 작업을 수행한다. 이때 이 대기 공간을 구현하기위해 Queue라는 자료구조를 사용하였다. Queue의 구현에있어 array를 사용할 경우 array size를 정함에 있어 모호함이 있다. 이러한 문제를 야기하지 않도록 linked list를 이용하여 Queue 자료구조를 사용하였다. Queue의 운용을 위해서는 5가지의 함수를 사용하였다.

Queue_Init	초기에 Queue 자료구조 생성
Queue_Enqueue	Queue에 node 추가
Queue_Dequeue	Queue에 node 제거
Queue_Front	Queue 최상단에 있는 node 반환
Queue_IsEmpty	Queue가 empty 인지 판단

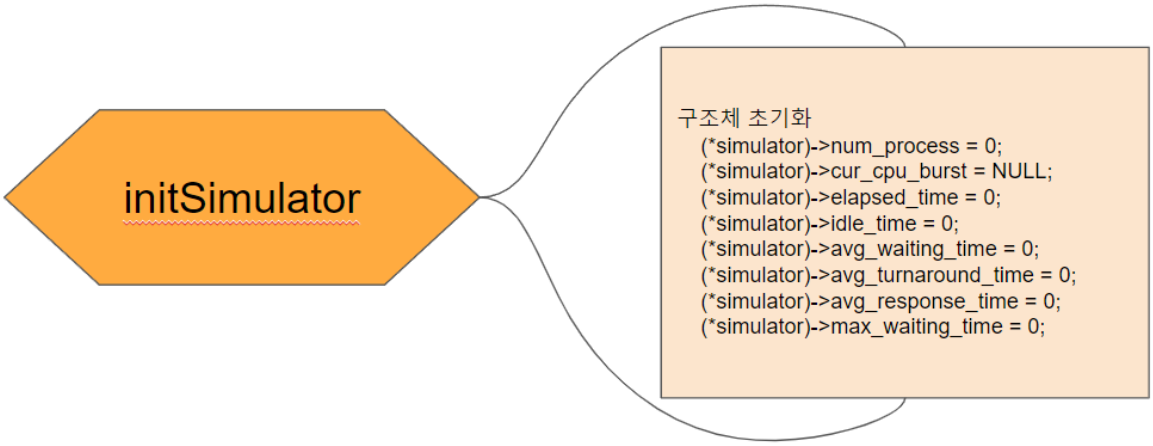
FCFS와 RR의 CPU scheduleing 은 ready_queue에 먼저 들어오는 프로세스 순서로 작업을 수행한다. 즉 FIFO 특징을 가지고 있으므로 ready_queue의 구현에 있어 상단에서 구현한 Queue 자료구조를 사용하였다. 하지만 SJF, SRTF, NON_PREEMPTIVE_PRIORITY 스케쥴러의 운용에 있어서 ready_queue는 FIFO의 성질을 갖지 않는다. SJF와 SRTF는 queue 내부에서 프로세스의 남아있는 burst time을 기준으로 재정렬 되고, Non_preemptive_priority는 우선순위를 기준으로 재정렬 된다. 이러한 기준을 반영하기 위해 Heap 자료구조를 구현하였다. 이는 Queue자료구조에서 우선순위에 따라 내부 배치가 재정렬 되므로 코드 내에서 PriorityQueue라는 명칭을 사용하였다. PriorityQueue에 운용에 있어서 5개의 함수를 사용한다.

Priority_Init	초기에 PriorityQueue 자료구조 생성
Priority_Enqueue	PriorityQueue에 node 추가
Priority_Dequeue	PriorityQueue에 node 제거
Priority_Top	PriorityQueue 최상단에 있는 node 반환
Priority_IsEmpty	PriorityQueue가 empty 인지 판단

PriorityQueue는 complete binary tree 구조를 갖고 있으며 Priority_Enqueue 시에는 정해진 기준에 따라 Enqueue를 시행한다. 가령 SJF 스케줄러의 경우 burst_time이 가장 적은 프로세스가 ready_queue의 상단에 와야 하므로 Priority_Enqueue 를 수행하면 burst_time이 가장 적은 프로세스가 tree의 최상단에 오게 된다. NON_PREEMPTIVE_PRIORITY 스케줄러의 경우 프로세스의 Priority에 기반하여 ready_queue를 재정렬 하게 되며, Priority가 가장 작은 프로세스가 tree의 최상단에 오게 된다.

II. Simulation

1. Simulator

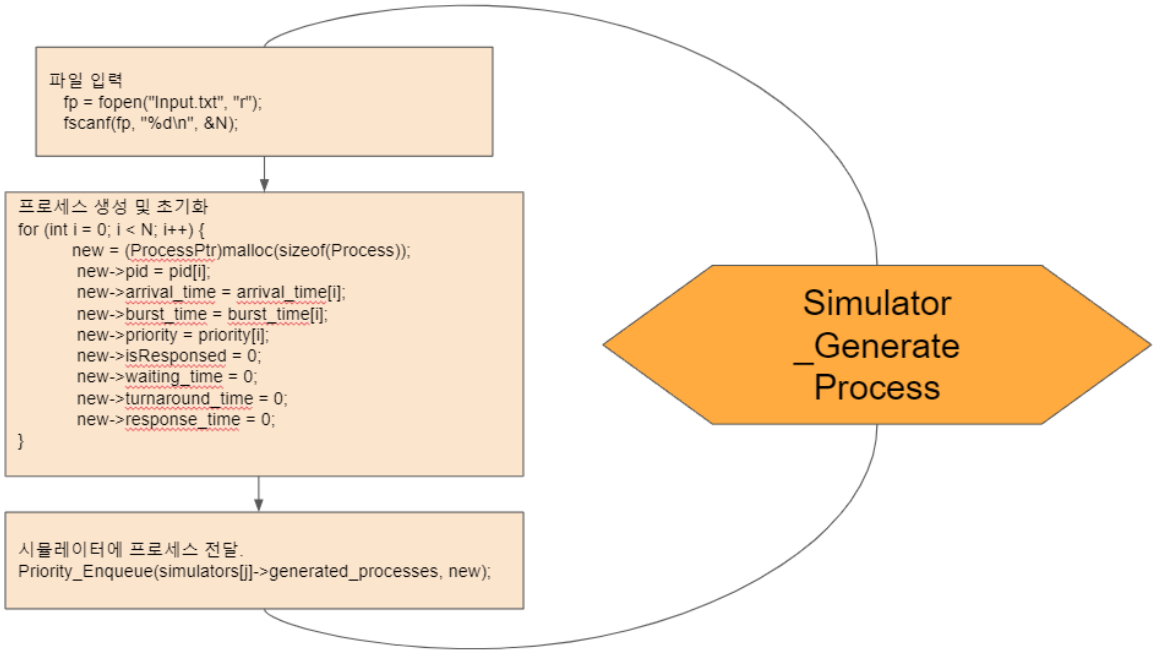


input.text를 read 해서 프로세스를 만들기 전에 각 알고리즘별 simulator를 미리 시작시켜 놓는다. 구조체 변수들을 초기화 한 다음, 알고리즘 FCFS, SJF, SRTF, RR, Nonpreemptive priority 각각은 플래그를 가져 해당 알고리즘이 필요로하는 process 변수들을 get한다.

```
initSimulator{

구조체 초기화;

알고리즘별 필요한 queue 종류 설정{
FCFS;
SJF;
SRTF: priority queue에서 남은 시간 길이 비교->reqdy_queue;
Nonpreemptive priority: priority queue에서 priority 비교->ready_queue;
RR: queue에서 일정 시간이 지나는 대로 실행을 멈추고 ready_queue로;
}

모든 프로세스별 도착시간 처리;

}
```

이들은 프로그래머가 정한 순서대로 배열되어 이후 시뮬레이터가 종료되며 원하는 순서대로 출력된다.

2. Process

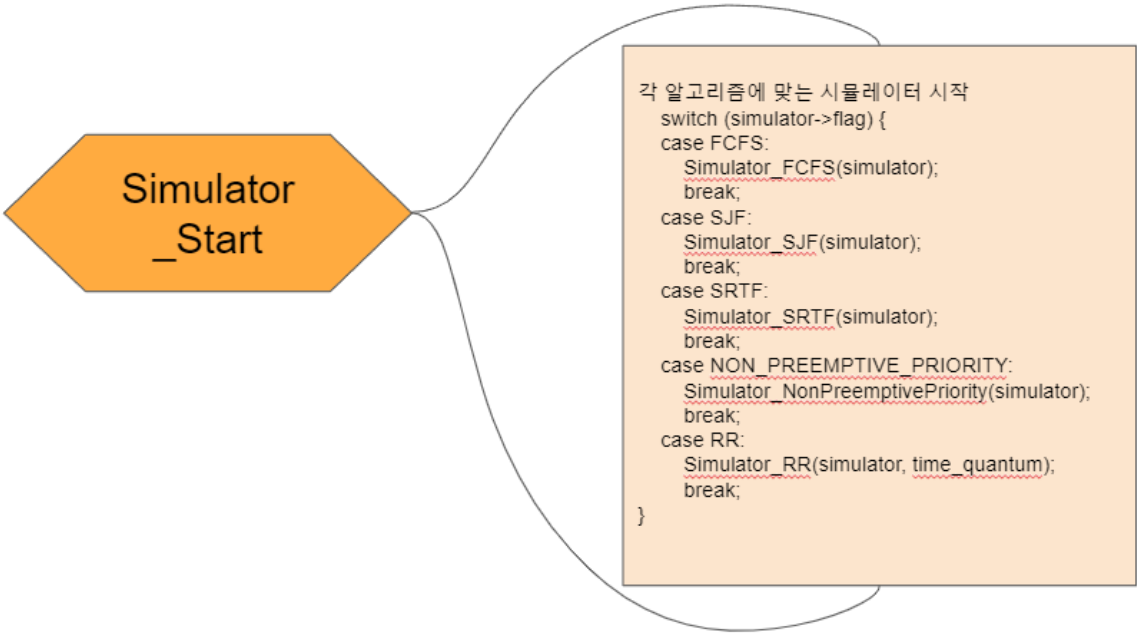


Simulator 는 미리 열려 있는 상태이므로그 대상인 Process가 필요하다. 이를 위해 Datastructure.h에 queue와 함께 Process 구조체를 정의했다. 실제 텍스트 파일을 읽어 process를 생성할 때는 Response bit, waiting_t, turnaround_t, response_t를 초기화하고 프로세스 개수 (N)에 따라 0~n-1까지의 프로세스들을 저장 및 초기화했다. 이후 Priority_queue()함수로 queueing하여 simulator가 프로세스들을 각 상황에 맞게 받도록 해 본격적인 scheduling algorithm이 돌아갈 준비를 해 둔다.

```
typedef struct _Process{
프로세스 id;
cpu 서비스 시간;
도착 시간;
우선순위;
응답 여부(0 or 1);
waiting_time;
turnaround_time;
response_time;

}
```

3. Start simulation & Evaluation

알고리즘별 simulation을 시작하고 gantt chart, Time 등 evaluation 대상들을 print한다.



1) 알고리즘별 simulation 시작-flag 설정으로 구현

```
Simulation_Start{
FCFS;
SJF;
SRTF;
RR;
Nonpreemptivepriority;
}
```

2) CPU scheduling algorithms 코드 핵심부분

각 알고리즘들은 ReadyQueue load, process load, waiting in queue, CPU burst 함수들로 이루어진다. 먼저 ReadyQueue load 함수는 해당 알고리즘이 사용할 Queue를 load하여 대기 중인 process, 처리 될 process, 종료된 process들을 나눠서 처리한다. 다음으로 Process Load 함수는 ready queue에서 process를 가져와 CPU 자원을 할당한다. Waiting in queue 함수는 대기중인 프로세스들의 변수를 조정하고, 마지막으로 CPU burst

함수에서 모든 프로세스의 남은 burst_time 이 0이 될 때 까지 진행되며 끝에 evaluation 함수 call을 통해 결과를 print한다.

FCFS

```
FCFS{
일반 Queue;

ReadyQueue Load;;
Process Load;
Waiting in Queue; // waiting processes들의 time 증가
CPU Burst{
turnaround time ++, burst time–, Response bit = 1;
간트차트 print;

  Evaluation{
  average 시간 계산;
  print time;
  }
};

}
```

SJF

```
SJF{
Priority Queue; // priority = burst time

PriorityReadyQueue Load;
Process Load in Priority;
Waiting in Priority Queue; //

CPU Burst;

}
```

SRTF

```
SRTF{
Priority Queue;

PriorityReadyQueue Load;
Process Load in Priority;
Waiting in Priority Queue; //

CPU Burst in preemptive; // CPU 할당이 preemptive하게 이루어지므로

}
```

RR

```
RR(maxtime){
일반 Queue;

ReadyQueue Load;
Process Load;
Waiting in Queue;

CPU Burst in RR(&time quantum, maxtime); // time quantum 처리.
//time quantum<=maxtime

}
```

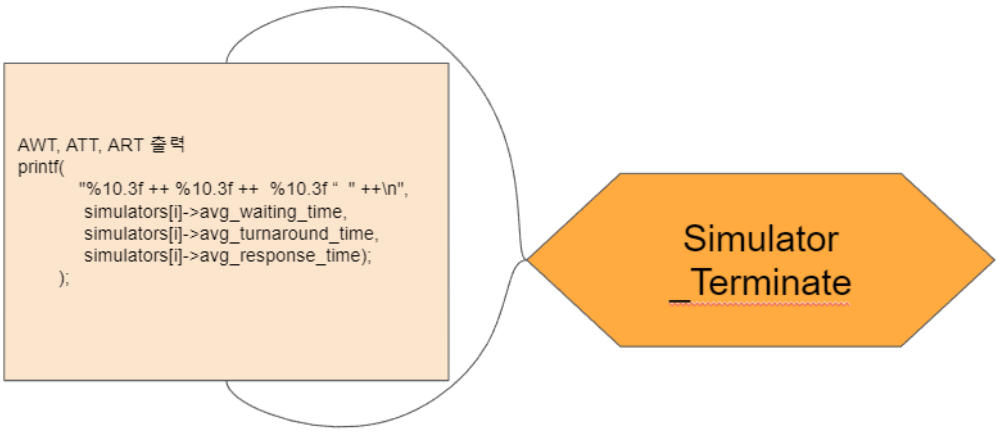
Nonpreemptivepriority

```
Nonpreemptivepriority{
Priority Queue;

PriorityReadyQueue Load;
Process Load in Priority;
Waiting in Priority Queue;

CPU Burst;
```

```
}
```

4. Terminate simulation & print summary

CPU scheduling algorithm의 성능 비교를 쉽게 할 수 있도록 Terminate 함수에는 아래와 같이 평균값들을 출력하고, simulator를 종료했다. 이 또한 algorithm별로 flag를 설정해 구현했다.




```
Simulation_Terminate{
FCFS;
SJF;
SRTF;
RR;
Nonpreemptivepriority;

결과 Summary로 print;
}
```

Conclusion: 실행 결과 및 소감

I. 실행 결과

프로세스의 인풋으로 다음의 text파일을 입력하였다.

Input - 메모장

파일 편집 보기

```
6
P0 0 4 3
P1 1 2 2
P2 2 3 1
P3 4 1 5
P4 7 5 2
P5 8 3 4
3
```

이에대한 콘솔 결과는 다음과 같다. 맨 처음 입력받은 프로세스 정보를 확인할 수 있게 프로세스 정보를 출력한다.

C:₩Users₩samsung₩Documents₩2022 상반기₩운영체제₩Term Project₩7조_TermProject₩OS_TermProject.exe

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++   PID   ++  CPU_BURST_TIME ++  ARRIVAL_TIME  ++  PRIORITY  ++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++      0  ++              4  ++             0  ++         3  ++
++      1  ++              2  ++             1  ++         2  ++
++      2  ++              3  ++             2  ++         1  ++
++      3  ++              1  ++             4  ++         5  ++
++      4  ++              5  ++             7  ++         2  ++
++      5  ++              3  ++             8  ++         4  ++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

이후 스케줄링 결과에 대한 간트 차트를 출력한다. 한칸당 시간단위 1을 상징하며 칸 내부에 적혀있는 숫자는 프로세스 번호를 뜻한다.

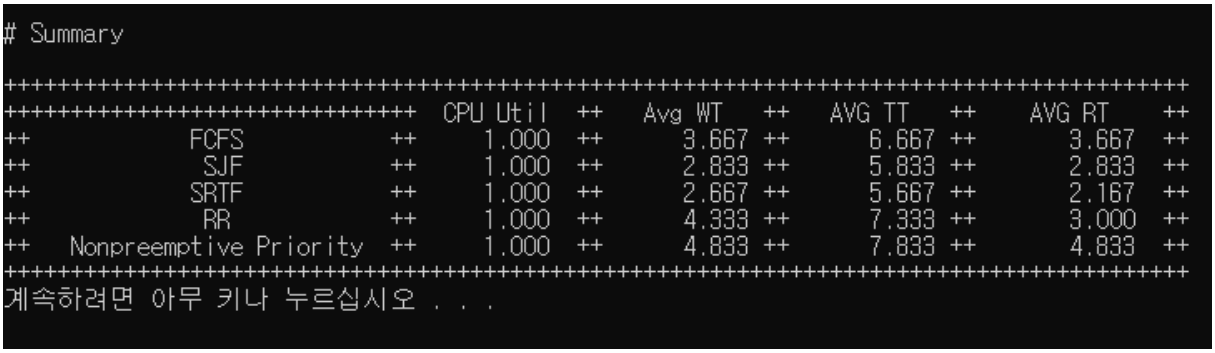
```
# FCFS Algorithm

[     0 ][     0 ][     0 ][     0 ][     1 ][     1 ][     2 ][     2 ][     2 ][     3 ]
[     4 ][     4 ][     4 ][     4 ][     4 ][     5 ][     5 ][     5 ]
-> Simulation End.
```

각 프로세스에 대한 Waiting time, Turnaround time, Response time을 출력한다. 또한 각각에대한 Average time을 계산하여 출력한다.

```
==================================================
(PID : 0)
Waiting time : 0
==================================================
(PID : 1)
Waiting time : 3
==================================================
(PID : 2)
Waiting time : 4
==================================================
(PID : 3)
Waiting time : 5
==================================================
(PID : 4)
Waiting time : 3
==================================================
(PID : 5)
Waiting time : 7
==================================================
Average waiting time : 3.667
==================================================
(PID : 0)
Turnaround time : 4
==================================================
(PID : 1)
Turnaround time : 5
==================================================
(PID : 2)
Turnaround time : 7
==================================================
(PID : 3)
Turnaround time : 6
==================================================
(PID : 4)
Turnaround time : 8
==================================================
(PID : 5)
Turnaround time : 10
==================================================
Average turnaround time : 6.667
==================================================
==================================================
(PID : 0)
Response time : 0
==================================================
(PID : 1)
Response time : 3
==================================================
(PID : 2)
Response time : 4
==================================================
(PID : 3)
Response time : 5
==================================================
(PID : 4)
Response time : 3
==================================================
(PID : 5)
Response time : 7
==================================================
Average response time : 3.667
==================================================
-> Execution time: 18
-> CPU Utilization: 1.059
-> Average waiting time: 3.667
-> Average turnaround time: 6.667
```

이후 순차적으로 SJF, SRTF, Round Robin, Non-Preemptive Priority에 대한 알고리즘 출력을 반복한다.

최종적으로 Summary를 출력한다.



II. 소감

강성묵: 여러 사람과 하나의 프로젝트를 진행해 보는 좋은 경험이었습니다. 혼자서 진행했다면 해낼 수 없는 프로젝트를 함께하는 것을 통해 한층 수월하게 해낼 수 있었습니다. 하지만 마냥 쉽지 만은 않았던것 같습니다. 여럿이서 함께하기 때문에 소통이 정말 중요하다는 것을 깨닫게 되었습니다. 옛날부터 개발자에게 협업 능력이 중요하다고 들어왔는데 그게 어떤 식으로 중요한지에 대해 피부로 느낄 수 있었습니다. 또한 복잡한 프로젝트일수록 flow chart 를 그리는게 중요하다고 깨달았습니다. 지금까지 프로젝트를 진행하면서 flow chart를 그릴 만큼의 큰 프로젝트를 해보지 않아서 여지껏 flow chart는 필요하지 않다고 생각을 했는데, 방향을 잃었을때 무엇을 구현해야하는지를 명확하게 보여주고 점점 복잡해 지는 코드를 간소화 하여 큰그림을 잃지 않게 해주는 좋은 도구였습니다. 본 경험을 통해 한층 더 성장할 수 있었습니다. 감사합니다.

남정화: 전반적으로 좋은 경험이었습니다. 평소 코딩 및 알고리즘 공부가 필요하다고 늘 생각은 했지만, 이번 일을 계기로 생각보다 몸이 알아서 먼저 하고 있을 것 같습니다. 결과가 나쁘지는 않지만 100% 만족은 아니라서 아쉬움이 남네요. 이전에 다른 형태의 여러 팀 활동을 진행해 본 경험이 있어서 자만했는지, 본인의 지식이 이번 과제 진행에 있어서 부족하다는 것을 알고도

철저한 수행 계획을 세우지 않고 막판 스퍼트를 낼 생각으로 진행했기에 더욱 그런 것 같습니다. 이번 Term Project 과정을 한 번 훑어보면 여러모로 본인의 지식 부족 뿐만 아니라 일을 처리하는 것에 있어서 방법론적으로 부족함이 많이 보입니다. 확실히 컴퓨터학과의 Term project는 이전에 진행했던 것들과 다르다는 것을 느꼈습니다. 사람 대 사람의 소통보다 사람 대 컴퓨터의 소통이 주가 되니까 그런 것 같네요. 아무튼, 깊은 바다에 한 번 빠지고 나서야 수영이 중요하다는 것을 깨닫는 것 처럼, 이번 term project는 제게 의미있는 기억이 되었습니다.

전병우: 전공 과목을 수강하면서 협업을 하는 프로젝트는 거의 없었는데, 이번 기회를 통해 소중한 경험을 할 수 있어 좋았습니다. 프로젝트 진행에 대한 회의부터 알고리즘에 대한 상호 피드백까지 거치면서 실무 중심의 테스크에서는 개발 능력 뿐만 아니라 협업 능력 또한 굉장히 중요하다는 것을 깨달았습니다. 특히, 좋은 개발자라면 단순히 코드를 작성하는 것을 넘어서 동료들이 읽기 쉽도록 해야함을 새로 알게 되었습니다. 좋은 코드란 무엇인지에 대해 재정의해보면서 알고리즘에 집중했던 지난날에서 좀 더 성장할 수 있었습니다. 힘들었지만 협업 가능한 개발자로 발돋움할 수 있어 가치있고 보람찬 시간이었습니다.